Technische Universität Berlin
Fakultät für Elektrotechnik und Informatik
Lehrstuhl für Intelligente Netze
und Management Verteilter Systeme

Virtual Network Management

vorgelegt von
Gregor Schaffrath (Dipl.-Inform.)
aus Neunkirchen

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

Promotionsausschuß:

Vorsitzender:	Prof. Dr.-Ing. Alexander Raake, TU Berlin
Berichtende:	Prof. Anja Feldmann, Ph. D., TU Berlin
Berichtender:	Prof. Laurent Mathy, Ph. D., University of Liège
Berichtender:	Dr. Olaf Maennel, Loughborough University

Tag der wissenschaftlichen Aussprache: 2. November 2012

Berlin 2012
D83

Ich versichere von Eides statt, dass ich diese Dissertation selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Datum Gregor Schaffrath

Abstract

The Internet has created many possibilities for communication and collaboration in a globalized world. Distributed systems have become widespread over the past decade, and are used for an ever growing set of applications and services. Many business models and economic processes are nowadays relying heavily on the continuously expanding set of operational networks. We expect this trend to continue, as data and processes move more and more out of localized infrastructure into globally accessible Cloud environments.

At the same time, however, its success has become a major problem to the Internet. Its architecture fails to address today's challenges. Its central role in modern economics and the number of actors which need to agree raise the stakes and costs for changing its core [64]. In fact, only the danger of an imminent collapse is expected to provide sufficient incentive to introduce architectural changes needed in the Internet [24]. Virtual networks (*VNets*) are a promising approach to overcome this 'ossification' by abstracting services from infrastructure and isolating network domains. In a future of virtualized cloud environments, they can, e.g., support on-demand service deployment where data and services are accessed via custom protocol stacks.

Still, like the Internet, VNets form a distributed system facing the problems of management in an environment with economic tussles which can impact on technical solutions [50]. A major challenge lies thereby in developing feasible management concepts which capture the concerns and interests of a plethora of actors. We expect competition to go beyond pricing and to involve tussles over service level abstractions, and mechanisms which may influence control over virtual resource mappings and connected benefits. These are likely to affect the technical solution, if the set of underlying assumptions (e.g., on information exchange and collaboration) leave insufficient flexibility in the required interfaces. Yet, current Cloud environments still mainly focus on host resources and available VNet architectures for testbed environments fail to address such requirements of operational networks.

In this thesis, we explore how to jointly manage production VNets and Cloud resources in multi-provider environments. Due to the importance of allowing for economic conflicts, we take a minimalistic approach on collaboration and information exchange assumptions. In the design of our solutions, we consider different realizations and contexts for their application. Our focus is on the applicability of our concepts to actual networks. We therefore implement them in a prototype both to gain insight into real world issues as well as to evaluate the feasibility of our design. We extend this to simulation whenever larger scales are required.

We first develop an architecture framework to manage VNets in a distributed multi-provider context. We identify roles and tasks based on network management and design interfaces supporting a variety of realizations. We propose a resource description language (*RDL*), which enables communication both between providers

and customers about requirements, and sites of, e.g., a substructured provider about current state of its infrastructure. Customers may or may not be interested in detailed properties of their VNet and over specification of such details can impact on a provider's benefits. Our RDL therefore supports different levels of abstractions and allows for both specific and vague descriptions.

We ask the question on how to dynamically place and manage resources. We formalize the problem of mapping virtual to infrastructure resources as Mixed Integer Program (MIP) which allows to calculate both optimal and approximate embeddings. This facilitates both a low-effort exchange of optimization objectives and leverages the use of highly optimized mathematical solvers which can give quality guarantees even for approximate solutions. In this, we explicitly consider migrations with respect to their cost. Moreover, we also propose a competitive online algorithmic approach to manage server placement in a scenario where a new service without known usage patterns is deployed.

In summary, we propose a structured, flexible approach to manage VNets in a distributed environment across different abstraction levels and administrative domains. Our concepts have been implemented and tested in a distributed prototype environment, managing actual network resources. They can provide insights into and guidance for how future VNets can be managed to enable a diversified network environment, in which resources can be leased for experimentation and on-demand deployment of customized networks.

Zusammenfassung

Das Internet ist zweifelsfrei eine der einflußreichsten Errungenschaften der letzten Jahrzehnte. Es eröffnete viele neue Möglichkeiten der Kommunikation und Kollaboration in einer globalisierten Welt, in der verteilte Systeme und Applikationen immer weiter an Bedeutung gewinnen. De facto sind viele Geschäftsmodelle und Prozesse ohne die stets wachsenden Kommunikationsnetze nur noch schwer vorstellbar. Im Rahmen der derzeitigen Tendenz, immer mehr Daten und Prozesse in global verfügbare Cloud-Infrastrukturen zu verlegen, erwarten wir, daß Kommunikationsnetze in der Zukunft eine immer zentralere Rolle spielen werden.

Während sich das Internet immer größerer Beliebtheit erfreut, stellt aber eben jener Erfolg ein Problem für seine Fortentwicklung dar. De facto trägt seine Architektur vielen modernen Anforderungen keine Rechnung. Seine zentrale Rolle in moderner Ökonomie, sowie die Zahl der betroffenen Parteien, welche sich zu Änderungen verständigen müssen, erhöhen aber Risiken und Kosten für Innovationen in seinem Kernnetz [64]. Bereits jetzt scheint nur noch ein unmittelbar bevorstehender Kollaps ausreichend Anreiz für die Umsetzung benötigter architekturaler Korrekturen zu bieten [24]. Virtuelle Netze (*VNetze*) abstrahieren Dienste von Infrastruktur und isolieren Domänen verschiedener koexistierender Netze. Sie stellen somit einen vielversprechenden Ansatz zur Überwindung dieser 'Verknöcherung' dar. In einer Zukunft virtualisierter Cloud-Umgebungen ermöglichen sie spontanes Ausrollen von innovativen Diensten, in denen Daten und Prozesse durch angepaßte Protokolle im Kernnetz unterstützt werden.

Allerdings teilen VNetze und das Internet Verwaltungsprobleme verteilter Systeme, in denen ökonomische Konflikte Auswirkungen auf technische Umsetzungen haben können [50]. Eine besondere Herausforderung liegt dementsprechend in der Entwicklung umsetzbarer Verwaltungskonzepte, die den Interessen und Bedürfnissen der Vielzahl Beteiligter Rechnung tragen. Wir erwarten, daß sich der Wettbewerb nicht nur auf Fragen der Preisgestaltung beschränken wird, sondern daß er ebenfalls Ressourcenbeschreibungs- und Ressourcenallokationsaspekte einbeziehen wird. Konflikte in diesen Bereichen werden wahrscheinlich technische Lösungen beeinflussen und Probleme schaffen, wenn den Mechanismen zugrunde liegende Annahmen (z.B. bzgl. des Informationsaustausches und der Zusammenarbeit) zu wenig Flexibilität in den Schnittstellen lassen. Bislang beschränken sich Cloud-Umgebungen noch überwiegend auf Host-Ressourcen, und bestehende VNetz-Architekturen für Testumgebungen tragen den Anforderungen wirtschaftlicher Umgebungen ungenügend Rechnung.

In dieser Arbeit betrachten wir die Frage der vereinheitlichten Verwaltung von Cloud-Ressourcen und VNetzen in Szenarien mit mehreren Anbietern. Wir identifizieren Aufgaben und Rollen in Hinblick auf die Verwaltung solcher Netze und entwerfen Schnittstellen, welche flexibel diversifizierte Umsetzungen unterstützen.

Wir erstellen eine Resourcenbeschreibungssprache, welche sowohl in der Kommunikation von Anforderungen zwischen Kunden und Anbietern, als auch in der Kommunikation von Infrastrukturzuständen zwischen z.B. Abteilungen eines untergliederten großen Infrastrukturanbieters einsetzbar ist. Diesbezüglich ist wichtig, daß Kunden ein unterschiedliches Interesse bzgl. bestimmter Detailbeschreibungen haben können, und sich Überspezifikation auf Gewinne des Anbieters auswirken kann. Unsere Beschreibungssprache läßt daher sowohl spezifische, als auch vage Beschreibungen auf verschiedenen Abstraktionsebenen zu.

Wir stellen weiterhin die Frage der dynamischen Ressourcenplatzierung und Ressourcenverwaltung. Wir formalisieren das Problem der Abbildung virtueller auf Infrastrukturressourcen als mathematisches Programm, welches die Berechnung sowohl optimaler, als auch angenäherter Ergebnisse erlaubt. Diese Formalisierung ermöglicht sowohl einen leichten Austausch der Zielfunktionen, als auch die Nutzung optimierter mathematischer Problemlöser, die auch für approximative Lösungen Qualitätsgarantien geben können. Wir betrachten hierbei explizit auch Migrationen in Hinblick auf ihre Kosten. Desweiteren entwickeln wir einen kompetitiven onlinealgorithmischen Ansatz zur Migration eines Servers bei unzureichender Kenntnis einer zu erwartenden Nutzung.

Zusammenfassend entwickeln wir einen strukturierten, flexiblen Ansatz zur VNetz-Verwaltung auf verschiedenen Abstraktionsebenen in verteilten Umgebungen mit mehreren Anbietern. Unsere Konzepte wurden in in einer verteilt angelegten Prototyp-Umgebung, in der sie tatsächliche Netzressourcen verwalten, umgesetzt und getestet. Sie erlauben Einsicht in die Fragestellung, wie zukünftige VNetze für diversifizierte Netzumgebungen verwaltet, und Ressourcen zur Erprobung oder schnellen Umsetzung innovativer Netze gebucht werden können.

Pre-published Papers

Parts of this thesis are based on the following papers that have already been published.

Articles

Even, G., Medina, M., Schaffrath, G., and Schmid, S. Competitive and deterministic embeddings of virtual networks. *Distributed Computing and Networking Lecture Notes in Computer Science 7129* (2012), pp 106–121

Feldmann, A., Schaffrath, G., and Schmid, S. Cloudnets: Combining clouds with networking. *ERCIM NEWS 88* (December 2011)

Feldmann, A., Kind, M., Maennel, O., Schaffrath, G., and Werle, C. Network virtualization - an enabler for overcoming ossification. *ERCIM NEWS 77* (April 2009), pp. 21–22

International Conferences

Schaffrath, G., Schmid, S., and Feldmann, A. Optimizing long-lived cloudnets with migrations. In *Proceedings International Conference on Utility and Cloud Computing (UCC)* (Chicago, USA, November 2012)

Schaffrath, G., Schmid, S., Vaishnavi, I., Khan, A., and Feldmann, A. A resource description language with vagueness support for multi-provider cloud networks. In *International Conference on Computer Communication Networks (ICCCN)* (July 2012), pp. pp. 1–7

Arora, D., Bienkowski, M., Feldmann, A., Schaffrath, G., and Schmid, S. Online strategies for intra and inter provider service migration in virtual networks. In *Proceedings on Principles, Systems and Applications of IP Telecommunications (IPTComm) conference* (Chicago, USA, August 2011)

Workshops

Arora, D., Feldmann, A., Schaffrath, G., and Schmid, S. On the benefit of virtualization: Strategies for flexible server allocation. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)* (Boston, USA, March 2011)

Bienkowski, M., Feldmann, A., Jurca, D., Kellerer, W., Schaffrath, G., Schmid, S., and Widmer, J. Competitive analysis for service migration in vnets.

In *Proc. ACM SIGCOMM VISA Workshop* (New Delhi, India, August 2010), ACM, pp. 17–24

Schaffrath, G., Werle, C., Papadimitriou, P., Feldmann, A., Bless, R., Greenhalgh, A., Wundsam, A., Kind, M., Maennel, O., and Mathy, L. Network virtualization architecture: Proposal and initial prototype. In *Proc. ACM SIGCOMM VISA Workshop* (Barcelona, Spain, August 2009), ACM, pp. 63–72

Technical Reports

Arora, D., Bienkowski, M., Feldmann, A., Schaffrath, G., and Schmid, S. Online strategies for intra and inter provider service migration in virtual networks. Technical report, arXiv, March 2011. ID arXiv:1103.0966v1 [cs.NI]

Schaffrath, G., Schmid, S., and Feldmann, A. Generalized and resource-efficient vnet embeddings with migrations. *CoRR abs/1012.4066* (2010)

Schaffrath, G., Schmid, S., Grassler, J., Abarca, E., and Feldmann, A. A virtual network architecture prototype. Tech. rep., to appear

Contents

Chapter 1

Introduction

While the Internet is currently viewed as widely successful for some of its participants, namely the users and the service providers, e.g., Google, it also suffers from ossification in the underlying infrastructure. As pointed out by, e.g., Petterson et al. [96], the ossification has multiple causes. Among them is the fact that the Internet works quite well as it is and therefore Internet Service Providers (*ISP*s) have no incentive to change their ways. Moreover, ISPs suffer from a lack of business perspectives due to the predominant charging modi [49]. An additional complication is that traffic grows at a higher rate than the costs for the network equipment decreases.

As a consequence, there is on the one hand an ongoing debate within the ISP community if ISPs should become pure bit pipe providers or if they should offer value added services. Several ISPs (e.g., Deutsche Telekom [12]) are extending their business to, e.g., Cloud or television services. On the other hand, service providers find that some application support inside the network can help their applications and consider investing in network infrastructure, e.g., Google [23].

Indeed, over the last 20 years most of the innovations, e.g., novel applications, have taken place at the edge of the network while the core is almost untouched. However, we believe the time has come to support novel applications with services inside the network, e.g., via network based enablers, and to revisit the Internet architecture to add native support for security, mobility, and manageability, etc. In this context, clean-slate research seeks to overcome the limitation to small incremental steps in core network innovation [64].

Among the reasons for slow core network innovation is a tussle between reliability and innovation [50]. In fact, costs are seldom balanced by economic incentives for ISPs, resulting in a situation where only drastic problems (e.g., IPv4 vs. IPv6) seem sufficiently strong motivations for architectural changes [24]. This tussle is therefore firmly biased on the side of reliability. New enabling technologies fail to achieve traction across the majority of ISPs. An important challenge is not only the search for solutions and improvements to the Internet's many problems, but in how to actually deploy those solutions and re-balance the tussle between reliability and functionality.

Virtualization allows for large-scale multitenancy which lowers unit costs for individual tenants. This, and an increased flexibility in resource allocations form the basis of potential CAPital and OPerational EXpenditure (CAPEX/OPEX) benefits, as can be observed in the context of Cloud hosting [6, 25]. Moreover, the isolation property of virtualization offers an approach for the required co-existence of reliable environments and environments with extended functionality [24]. To this end, network virtualization extends virtualization from the host domain to entire networks sharing a commong physical infrastructure. It is not to be confused with current technologies such as Virtual Private Networks (*VPNs*), which merely provide traffic isolation: full administrative control, as well as potentially full customization of the virtual networks (hereafter: *VNets*) are also required. This also means that non-IP networks may then run alongside the current Internet realized as one future VNet. Indeed, each of these virtual networks can be built according to different design criteria and operated as service-tailored networks. It becomes possible to create core network support for mutually exclusive design goals, and to introduce them in parallel with legacy networks as the Internet. For example, one network may support full authentication and prevent spoofing by performing plausibility checks on routing core instances, another may be designed in support of anonymity or plausible deniability. The internet itself may be encapsulated in a VNet, or serve as underlying network for tunneled VNets.

Economic benefits are not limited to physical infrastructure owners. Service providers may use virtualization to limit financial risks: Physical infrastructure deployments are expensive and usually not an option for short term experimental setups. On the other hand, the success of Cloud computing [74, 75] shows that the leasing of virtual resources can be affordable and achievable for various time spans with a variety of different tariff models. This makes VNets an attractive option for service providers to, e.g., roll out experimental setups in realistic environments, to control change effects by regression tests [135], or to simply aquire cheap redundancy and spillover resources.

Indeed, in today's Internet, ISPs as well as service providers (e.g., Google) are continuously searching for opportunities to either increase revenue or to reduce costs by launching new services, investing in new technology or by decreasing operational costs. To understand the order of magnitude of the investment cost consider that in 2009, AT&T planned to invest 17–18 Bn $ in 2009 [4] compared to a revenue of 124 Bn $ in 2008 [5] and Deutsche Telekom invested 8.7 Bn € compared to revenues of 62 Bn € in 2008 [3]. Thanks to increased resource sharing, even a modest reduction in the investments of, say, 1% can result in several millions of savings per year.

However, an approach to manage VNets is required to leverage all of the abovementioned advantages. Moreover, such an approach needs to consider and account for technical and economic competition in the real world [50]. We expect new business concepts, e.g., based on reselling of infrastructure for world wide spanning virtual infrastructures, to incite the clash of a variety of interests in every domain.

This thesis addresses the question of VNet management. In the following sections, we name important challenges in this domain, we outline our approach and contributions, and give an overview over the remainder of the document.

1.1 Challenges

Like Cloud hosting, VNets can be used for on-demand service deployment. Moreover, world wide spanning VNets are likely to involve multiple providers. ISPs and other stakeholders are likely to be involved in competition on many levels. We therefore aim at the minimization of economic tussles in the technical domain in all our efforts. Approaches and concepts are devised to allow for a variety of different and competing management approaches. In order to achieve this flexibility, we limit collaboration assumptions (e.g. information flows between different players) as far as possible. We argue that a larger set of assumptions — e.g., more extensive collaboration and information exchange — may allow for multilateral optimizations, but is unlikely to break the approaches as such. Moreover, we aim to devise approaches in a way that allows actors to focus on aspects of their interest scopes whenever possible.

A major challenge lies in the alignment of framework elements with both technical and business requirements. Interfaces must be designed to facilitate inter-provider cooperation and yet allow control over business secrets, such as information about owned infrastructure topology. Tasks and required knowledge need to be identified and matched against responsibilities of defined roles. A framework forcing actors to e.g., publish business secrets or lacking technical flexibility to adapt it to different management approaches will likely fail from start. Likewise, the roles should facilitate focusing on specific tasks in order to allow for economic concepts, such as outsourcing.

If multiple actors are to cooperate on the partially or fully automated provisioning of virtual topologies, they require a description language to describe and communicate topologies as well as requirements. Moreover, large providers with a substructured infrastructure may want to gather information about the use of their resources across sites. Therefore, the resource description language (*RDL*) should be able to describe not only requirements, but also assignments of virtual to infrastructure resources. It is important to keep such a language adaptable to respective contexts: Competition between providers is likely not only to involve tariffs, but also service levels and descriptions. Customers may or may not be interested in details on an arbitrary mixture of abstraction levels. Providers may respond by adapting and intermixing these abstraction levels in offers and their communication with the customers. Furthermore, the levels of specificity in requests interfere with a provider's flexibility in resource allocation optimization, and thereby impact on his benefits. An RDL therefore must not only be conceptually extensible, but must also allow every actor to focus on his or her core interests. It must allow for vagueness as well as

specificity. Moreover, it should facilitate translation of requirement descriptions into forms which facilitate the calculation of resource assignments (*mapping*).

Networks are composed of many different types of resources. Moreover, the allocation of virtual resources is not necessarily limited to physical resources of the same type. Finding an optimal mapping with constraints on both virtual nodes and links is known to be NP-hard [22]. Therefore, both optimal and approximate approaches need to be explored for different scenarios and scopes. In this task it is important to facilitate the handling of actual network topologies including, e.g., shared resources and multi-homed links in both virtual and physical topologies.

VNets enable dynamic and transparent provisioning. Resources of a VNet may be transferred (migrated) to different parts of the infrastructure, e.g., to adapt to changed load levels. One major question however is on how to leverage this advantage in scenarios where no information about future usage patterns are available. Competitive online strategies are required to guarantee competitive performance, e.g., for novel services.

Whenever possible, we aim to create real world applicable mechanisms and approaches and to limit the level of simplification as much as possible. This applies both to components managed and interactions of managing entities. To this end, we implemented key components in a prototypical proof-of-concept implementation.

1.2 Approach and Contributions

In this thesis, we

- design an architecture for VNet management in multi-provider scenarios in terms of roles, tasks, and interfaces. The architecture design is based on both technical and economic considerations and allows for tussles. Granular roles with minimum assumptions on inter-provider cooperation allow for split-up of essential tasks into competing corporate entities, without barring optimizations in close cooperations between roles. Recursive roles allow for subcontracting scenarios, and we consider different approaches to end user terminal attachment in support of scenarios where not all ISPs are VNet-aware.
- devise a resource description language (*RDL*) for communication between the roles. The extensible RDL is able to describe real world networks, yet is based on a graph-like structure to facilitate the application of graph algorithms in the context of mapping. Its design considers the translation into full-fledged ontologies for automated reasoning, and allows for specification and omission of arbitrary descriptive details on arbitrary abstraction levels. This, in combination with black- and white-listing facilitate diverse degrees of specifity in support of customer foci and provider optimization interests.

- formalize VNet mapping in the form of a Mixed Integer Program (*MIP*) to leverage the use of optimized mathematical solvers to calculate optimal embeddings. Like the RDL, the MIP is able to operate on real world topologies and facilitates an easy exchange of optimization goals. We identify migration cost factors and include their consideration in the formulated MIP. We sketch further extensions and examine qualitative effects of approximations on its performance. Performance evaluations are based on both artificial and real world measured Rocketfuel [117] topologies.
- develop a provable competitive approach to manage migration of a server in single or multiprovider online scenarios without knowledge of future events. The approach decides on both time and destination of migrations where server access costs in correlation with the distance to optimal positions are to be weighted against migration costs. We sketch an extension for multi-server scenarios and evaluate the approach in a simulated time-zone scenario.
- implement a proof-of-concept prototype. The prototype is based on our architecture and RDL. It uses our MIP to map VNet requests, and is interconnected with the sister prototype site of our collaborating partner Docomo EuroLabs Munich. We evaluate multiple scenarios including multi-provider scenarios, recursive virtualization, and dynamic embedding based on usage conditions of a service running inside a VNet.

1.3 Structure of the thesis

In Chapter 2, we give an overview on the background. In Chapter 3, we present our architecture for distributed management of VNets across multiple infrastructure providers. Given that the different entities need to communicate about requirements and mappings, we next address the question of topological resource description in Chapter 4. In Chapter 5, we formalize the mapping problem problem itself and investigate into approximation trade-offs in Chapter 6. Competitive migration online decision algorithms are presented in Chapter 7. Chapter 8 presents our proof-of-concept implementation in terms of components and setup. We conclude in Chapter 9 with an outlook on future work.

Chapter 2

Background and Related Work

This chapter gives an introduction into the background and concepts which are used in this thesis. The first section explains the concept of virtualization and puts virtual networks into context with related concepts, including 'the Cloud' and 'Software Defined Networking' (*SDN*). The second section provides the background information regarding mixed integer programming. The background chapter concludes with a glossary of the most important terms.

2.1 Virtualization

This section begins by an introduction into the concept of virtualization. It gives a short overview over common host virtualization variants, and then puts Virtual Networking into context with other fields. Finally, we describe the concept of migration in the context of virtualiztion.

2.1.1 What is virtualization?

Virtualization as such is not a new concept. It describes the abstraction of parts of physical resources and entities into 'virtual' ones, which — in their respective context — are indistinguishable from their real counterparts. The idea was first sparked in the context of time sharing on large expensive mainframes. C. Strachey's propositions in [118] incited the development of mechanisms to create virtual resources (e.g., Virtual Memory) and hosts. The latter were designed to mimic a separate machine that was allowed to coexist with others of its kind, sharing underlying physical resources (now often referred to as '*substrate*'). These virtual hosts are managed and controlled by a so-called 'Virtual Machine Monitor' (*VMM*), which is nowadays often referred to as '*Hypervisor*'. For a short history of host virtualization, the reader is referred to a variety of summaries and surveys, e.g., [115], [106].

The most widely accepted requirements on virtualization has been proposed by Popek and Goldberg [98]. With a focus on host virtualization, they postulated three essential requirements:

1. "Any program run under the VMM should exhibit an effect identical with that demonstrated if the program had been run on the original machine directly." The only exception in his view is timing.
2. "A statistically dominant subset of the virtual processor's instructions are executed directly by the real processor."
3. "The VMM is in complete control of the system resources."

Not all requirements are always fullfilled by all mechanisms. For instance, the second requirement has been a problem on the x86 architecture for a time (see, e.g., [106]). However, these requirements point to the core characteristics of virtualization: Transparent resource sharing (abstraction) and domain isolation. The virtualized entity should behave like a physical entity in as many respects as possible.

2.1.2 Host virtualization

There are three basic types of host virtualization in use today: Full virtualization, paravirtualization, and container based virtualization.

2.1.2.1 Full virtualization

Full virtualization allows virtual hosts (or '*guests*') to run unmodified operating systems (*OS*). Among the first and most prominent fully virtualized machines was IBM's VM/370. One hypervisor runs in priviledged mode and has access to all instructions. Guests mostly run on the real CPU, but some crucial instructions are 'trapped' (i.e., generate interrupts handled by the hypervisor). Until around 2004, the X86 architecture did not support these traps in hardware (consult [106] for detailed explanation), barring virtualization mechanisms from fulfilling the second requirement. The additional ring used by hypervisors is commonly referred to as *hardware virtualization*. While the transparency of full virtualization is an advantage in terms of flexibility, its backdraw consists in high overhead: Even hardware virtualization involves many context switches when trapping abovementioned sensitive instructions. Among the most prominent contemporary virtualization mechanisms supporting full virtualization are KVM[19], VMWare[124], and recent versions of Xen[30].

2.1.2.2 Paravirtualization

Paravirtualization allows the use of other OSs, but requires these to be modified. Again, a hypervisor manages its guests running in priviledged mode, which in turn run mostly on the real CPU. In contrast to full virtualization however, sensitive instructions are not trapped. Instead, the guest's operating system is modified to

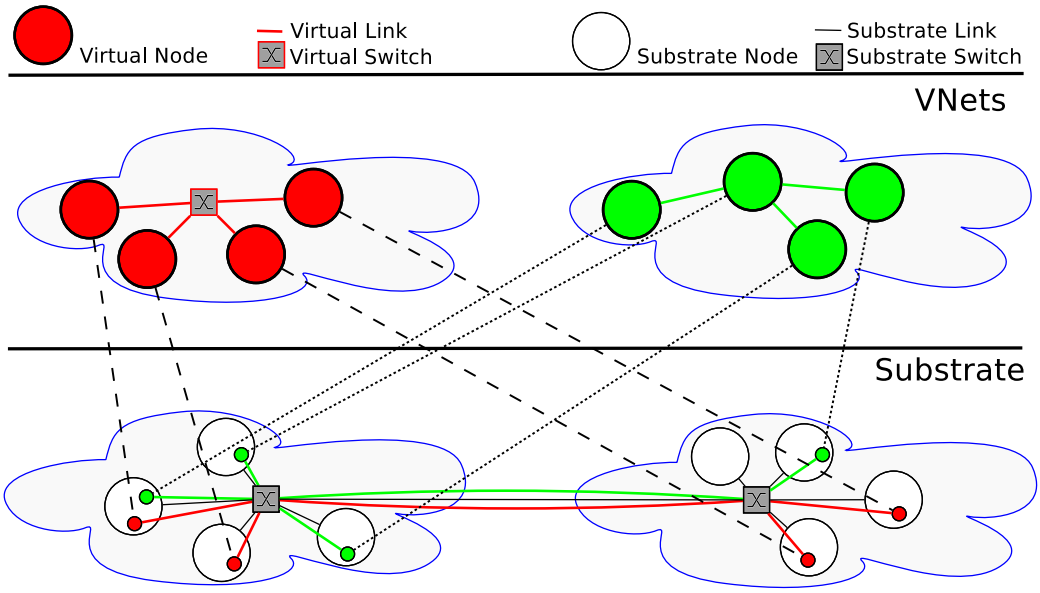


Figure 2.1: **Virtual Networks mapped onto a common substrate**

call routines of the host OS directly. While this obviously reduces flexibility with respect to guest OS choice, its benefits lie in performance advantages. Xen and User Mode Linux[133] were most prominent virtualization mechanisms that focused on paravirtualization for a long time.

2.1.2.3 Container based virtualization

Container based virtualization does not allow other guest OSs. Only a single machine with a single kernel runs on the physical machine. Virtualization consists of an isolation of the running user space processes and environments. This approach involves little to no overhead, as all processes send system calls directly to the only kernel running. However, it does not allow the instantiation of another kernel.

It is important to note that it is possible to a certain extent to run virtual hosts in virtual hosts. More specifically, CPU emulation may allow for (slow) full virtualization, and paravirtualized solutions may run in hardware virtualized environments. For more in-depth descriptions of host virtualization techniques, we refer the reader to surveys, such as [47].

2.1.3 Virtual networks

VNets consist of a combination of virtual nodes (*VNodes*, e.g. hosts, switches, routers) and links (*VLinks*). As pictured in Figure 2.1, multiple of such VNets can

be co-hosted on a shared substrate. Enablers for link virtualization are available on every network layer. For instance, time and frequency multiplexing can serve that purpose on the physical layer, VLANs[2] are an obvious choice on the link layer, and VPN connections via, e.g., OpenVPN[94] are common on the upper layers.

By including network components, such as routers and switches), and links, network virtualization extends the virtualization abstraction from hosts to entire networks. Most host virtualization mechanisms support the configuration of virtual network interfaces and even links inside one physical machine. In contrast, integrated management across real physical netwoks, comprising embedded switches and routers, is underdeveloped.

Around the year 2000, researchers tried to facilitate customization of networks. There were two major schools, which [43] provides an overview over. 'Programmable' networks considered networks as a programming problem. They advocate open interfaces on network equipment used by distributed running programs. In its context, e.g., the Genesis project ([56], [122]) aimed to implement networks in form of a distributed running CORBA[123] program. At the same time, so-called 'active' networks (AN) went even further by allowing transported packets to modify the network component's behaviour. Several projects (e.g., NetScript [108], SmartPackets(Kansas) [82]) aimed to facilitate remote installation of customized routines on routers. These in turn could be referenced by packet content on network routers. Some other (e.g., SmartPackets(BBN) [114], ANTS [55]) went as far as to allow packets to carry their own processing routines. Partly due to a lack of practicability in commercial production network environments (see, e.g., [100] for some AN security issues), these projects were not widely adopted.

Recently, customization of networks is revisited in the context of 'Software Defined Networks' (*SDN*) under more realistic assumptions. One its most prominent projects is OpenFlow [88]. OpenFlow switches consider bit pattern namespaces rather than protocols and protocol headers. Controllers are allowed to configure rules on the processing devices. The rules may apply to a specific nameset the controller has been assigned, and controllers may be structured in a hierarchy. Infrastructure owners deploying OpenFlow retain essential control over their assets. This aspect may play an important role in its success.

2.1.4 Migration

Migration is the process of moving a virtual entity to a different set of resources. Migration effects may be visible inside of the virtualized environment, but ideally are not (i.e., are *transparent*). Host migration usually involves several steps:

1. pausing (suspending) the virtual machine and serializing its state,
2. transferring the machine and its state to its new location,

3. possibly adapting either the machine or its new environment (e.g., w.r.t. interface associations between substrate and guest),

4. restarting (resuming) the machine.

Given the amount of data which needs to be transferred (e.g., RAM and virtual hard drive content), these migrations are usually connected to a temporary unavailability. So-called '*live migration*' aims to minimize this time interval to an extent where it becomes barely noticeable. It does not immediately suspend the virtual host, but rather starts by transferring a snapshot of it while it is still running. It then proceeds to send deltas of its state, until the difference between the location is small enough to proceed with the last delta as described above.

Until recently, most mechanisms supported live migration only in (almost always local) environments where source and destination substrate hosts shared access to virtual bulk data (HDD) resources. Recent extensions enable, e.g., KVM to perform live migrations over wide areas, as described in, e.g., [40].

If Links are not virtualized, VNodes may need to adapt to the changed network environment (e.g., w.r.t. IP addresses and routing). Transparency therefore is not always possible in environments where only host resources are virtualized. Virtual networks hide the real environment of VNodes. They thereby create the possibility to abstract these changes, and create a stable environment, where migration is handled entirely in the substrate.

Evidently, in such environments not only host, but also link resources need to be migrated. As network state is rarely connected to bulk data, these migrations are often effectuated in terms of reinstantiations and configuration changes on network equipment. The establishment of intermediate tunnels, as described by, e.g., VROOM [126], may help to reduce network interruptions.

2.2 Mixed Integer Programming

Many optimization problems may be formulated in mathematical form as so-called Linear Programs (*LP*) or Mixed Integer Programs (*MIP*). These may be solved by highly optimized solvers. This section first gives a definition of the terms Linear and Mixed Integer Programs. We then give an example in form of a Multicommodity Flow Problem (*MFC*) variant. We then describes basic techniques used in solving Linear Programs which are of relevance to this thesis, and finishes with a short discussion on solvers and solving runtime issues. For a more in-depth introduction on the topic, the reader is referred to standard literature, such as [53, 78, 134].

2.2.1 Linear Program

A Linear Program (LP) describes a linear optimization problem in the form of an *objective function* $\min c^T x$, and a set of constraints *constraints* $Ax \leq b$, for $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. This form is commonly referred to as *primal* of the problem.

The *dual* of an LP has the form of (*objective*) $\min b^T y$, and (*constraints*) $A^T y \geq c$, for $A \in \mathbb{R}^{m \times n}$, $y \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. While it is only a transformation of the *primal*, it often plays an important role in the solving process.

An Integer Linear Program (ILP), or Mixed Integer Program (MIP) is a program whose variables are partially (MIP) or fully (ILP) integer constrained (i.e., some or all $x_i \in \mathbb{Z}, y_j \in \mathbb{Z}$). An important difference is that pure LPs can be efficiently solved, which may not be the case for MIPs or ILPs. For instance, the Multi-Commodity Flow Problem (MCF) can only be solved in polynomial time when it can be relaxed to a pure LP [53].

2.2.2 Example: Multi-Commodity Flow Problem

The Multi-Commodity Flow problem consists of a capacitated transport network and multiple commodities which are to be transported across the given transport network. As an example, consider the simplified variant presented in [53], where the transport network is formally represented by a weighted directed (non-bidirectional) graph $G = (V, E)$ of vertices $v_i \in V$ and edges $(u, v) \in E$ of weight (capacity) $c(u, v) \geq 0$. Non-existing edges are implied to have 0 capacity, as are inverse directions of edges. In this example, we consider a graph in which $c(u, v) > 0$ implies $c(v, u) = 0$. An amount d_i of k commodities $K_i = (s_i, t_i, d_i)$ is to be transported as flows from source s_i to target destination t_i. Thereby, $f_i(u, v)$ symbolizes the flow of K_i, and $f(u, v)$ the aggregate flow of all K_i over the edge (u, v). Then, the problem of finding any feasible solution (no optimization objective) may be described as follows:

$$
\begin{aligned}
&\text{minimize } 0 \\
&\text{subject to } \sum_{i=1}^{k} f_i(u,v) \leq c(u,v) && \forall (u,v) \in E \\
&\sum_{v \in V} f_i(u,v) = 0 && \forall K_i \in K; \forall u \in V \setminus \{s_i, t_i\} \\
&f_i(u,v) = -f_i(v,u) && \forall K_i \in K; \forall (u,v) \in E \\
&\sum_{v \in V} f_i(s_i,v) = d_i && \forall K_i \in K \\
&f_i(u,v) \leq c(u,v) && \forall K_i \in K; \forall (u,v) \in E
\end{aligned}
$$

The *objective* of 0 renders any feasible solution optimal. The first constraint ensures that the sum of all commodity flows will never exceed available edge capacities. The

second ensures that flows are preserved in all intermediate vertices u (that are neither source, nor sink of the flow). To this end, all incoming flows with respect to u need to be counted as negative (third constraint). But since incoming flows in v would create bogus capacity in the opposite direction with respect to the first constraint, the last ensures that no flow is routed into a direction without capacity.

2.2.3 MIP solving steps

Software accepting MIPs (e.g., CPLEX[127], SCIP[128], Gurobi[129], SoPlex[131], lpsolve[130]) usually take several steps in solving. This subsection will focus on a few techniques that are of relevance in the context of this work. For detailed explanations, refer to abovementioned literature.

2.2.3.1 Presolving

As described in, e.g., [21], most solvers analyze the problem before attempting optimization. During the presolving step, they attempt to detect and to eliminate redundancies. This serves the purpose to reduce the problem and the connected computational complexity for the actual solving algorithms.

2.2.3.2 LP relaxation

One technique is to remove integer constraints and consider the pure LP, which can be efficiently solved by, e.g., the Simplex or Interior Points method. The linear optimum, however, is in no way guaranteed to be near a good or optimal integer solution. If the solution space's hull is concave, the integer solutions surrounding the linear optimum may be arbitrarily suboptimal. For this reason, the technique is often used only as a subroutine in another approach, such as branch and bound.

2.2.3.3 Branch and bound

This technique first relaxes the integer constraint and solves the pure LP. If the solution is not integer, it chooses one variable x_i of non-integer valuej $val(x_i)$ to branch the problem, by dividing the problem into two subproblems. One subproblem receives the additional constraint of $x_i \leq \lfloor val(x_i) \rfloor$, the other subproblem the constraint of $x_i \geq \lceil val(x_i) \rceil$. This procedure is repeated recursively with other $x_{i'}$, whenever a branch shows potential for better objective values, until the optimum is identified. The tree spanned by the different variable branching operations is often referred to as *decision tree*. Traversing the tree of possible branching decisions, the solver is said to *expand nodes*.

2.2.3.4 Cutting-plane

Integer solutions are also part of the solution space of the relaxed LP version of the problem. This technique adds constraints to cut non-linear optima from the convex hull of the MIP solution space. Again, it is usually not used as a standalone technique, but most often in combination with branch & bound. This combined approach is called *branch & cut*.

2.2.4 Solver runtimes

Optimal Linear and Mixed Integer Programs solutions can be calculated by highly optimized mathematical solvers. Using techniques like the ones described above, they search for integer solutions and compare them to found unconstrained solutions. More specifically, they usually consider the value of the Best Integer solution (BI) and Linear Bounds (LB). Most solvers output a quality guarantee GAP at runtime which is calculated as $GAP = \frac{BI-LB}{BI}$ and terminate when GAP falls below a configured threshold.

Even though solvers mostly rely on the same set of basic techniques, they exhibit major differences in runtime. For a comparison based on a standardized set of hard problems, see [80, 92].

The reason for this lies in the use of various heuristics to find the optimal solution, including heuristics for, e.g., questions as to how long to presolve problems, or when to invest additional effort to find intermediate integer solutions. Finding good cutting planes is a highly non-trivial problem. Moreover, a major influence factor is the order in which branching nodes are exanded (i.e., branching variables are selected, and when to explore which subtrees).

The software usually exposes most heuristic parameters configuration. Default heuristics are often considered a valuable business secret. This effectively turns most well performing commercial products into black boxes and makes it hard to reason about cause and effect. Unfortunately, runtime does not only vary over solvers and problems, but also over parameter sets, problem instances, and even the order in which constraints are formulated [80].

Even minor formulation changes can change runtimes drastically. Moreover, for reasons outlined in [81], effective parallelization of MIP solving is still a research question. This is reflected in the results of [80], which show that even one added or removed processing thread may change very good runtimes into very bad runtimes (and vice-verca). As a consequence, it is effectively impossible to predict which solver and parameter set will lead to which runtime. This limits generic validity of runtime evaluations, and allows identification of trends, but no definitive conclusions.

2.3 Glossary

Console: Out-of-VNet interface of nodes

Edge: Tuple $e = (u, v) \in E$ of vertices $u, v \in V$ of a graph $G = (V, E)$

Embedding: Realization of a VNet mapping.[1]

Graph: Tuple $G = (V, E)$ of a vertice set V and an edge set E

Mapping: An assignment of (or the process of assigning) virtual resources to substrate resources

Network Element (NE): Either a node or a link

Node: An active networking component (e.g., a host, router, switch, hub,...)

Link: A communication channel between two or more nodes

Substrate: The underlying infrastructure hosting a virtual network. The substrate is an underlay to the VNet

Substrate Link (SLink): A substrate link

Substrate Network Element (SNE): A substrate host or link

Substrate Host (SHost): A substrate host

Terminal: End user device, which can be attached to a network and may be mobile

Vertice: Element $u \in V$ of a graph $G = (V, E)$

Virtual Link (VLink): A virtual link

Virtual Network (VNet): A network of virtual nodes and virtual links. The VNet is an overlay to the substrate

Virtual Network Element (VNE): A virtual host or link

Virtual Node (VNode): A virtual node

[1] Please note that some parts of this thesis use the term embedding as synonyme for mapping, when the distinction doesn't change the meaning

Chapter 3

Architecture for Virtual Networks

In the introduction, we argue that VNets hold many opportunities for the introduction of innovative services. In some use cases, these VNets may need to span around the globe (e.g., to ensure QoS and custom protocol support on the entire connection to end user terminals). Expansion of network infrastructure is connected to risks [132], which make it unlikely that world wide VNets will be provided by a single ISP alone. Rather, it can be assumed that collaboration of providers and the leasing of infrastructure for reselling purposes will be in order. However, considering multi-provider scenarios adds considerably to the challenges to the management of infrastructure and the mapping of virtual networks: Economic tussles may spawn around responsibilities and authorities. Information availability (e.g., w.r.t. substrate topologies) and exchange may be restricted.

In this chapter we present a network virtualization architecture for a Future Internet, which we motivate through an analysis of business roles. In contrast to the GENI initiative [69] our goal is not to provide an experimental infrastructure but to see which players are necessary to offer virtual network based services to everyone.

The first section of this chapter will consider the abovementioned constraints and challenges in order to derive design guidelines for the following architecture design. The architecture will be presented in terms of roles and control plane. We discuss related work and conclude with an analysis of the architecture's flexibility with respect to forseeable tussles.

3.1 Challenges and Approaches

Currently existing architectures mostly target testbed environments. Testbed architectures, however, usually assume a single authority with overall knowledge. Production environments on the other side consist of a multitude of providers with extensive communication and cooperation constraints. Ceding configuration control of owned substrate devices to competitors or divulge one's own infrastructure topology is unlikely to be an option. The architecture must allow actors to keep their business secrets and control of their assets.

Control about resource placement (i.e., hosting contracts) is related to control about VNet provisioning benefits. Resource placement therefore seems likely to turn out a major economic battlefield: Some startups may attempt to establish themselves as single broker or reseller, whereas major ISPs will probably counter with their own brokers, cartels and extended services. It hence is important to allow for flexible grouping of management tasks with respect to actors without excessive impact on interfaces and the technical domain. Moreover, it must be possible to allow for subcontracting and substructuring of large providers.

We therefore design the architecture with the most restricted set of assumptions in mind. We argue that a relaxation of these assumptions may very well allow for optimizations, but is unlikely to break the framework as such. We therefore assume a strict abstraction between the virtual and the substrate domain. Embeddings are assumed to be transparent to customers and information flow between entities is assumed to be minimal (e.g., virtual network requests must contain requirements, but not necessarily the entire topology).

3.2 Roles

The major actors in current Internet are service providers (e.g., Google) and Internet Service Providers (ISPs). Hereby, an ISP offers customers access to the Internet by relying on its own infrastructure, by renting infrastructure from someone, or by any combination of the two. Service providers offer services on the Internet. In essence, ISPs provide a connectivity service, very often on their own infrastructure, even if they also lease part of that infrastructure to other ISPs. For example, AT&T and Deutsche Telekom are mainly ISPs while Google and Blizzard are SPs.

Despite this "dual-actor landscape" [63, 141], there are already three main business roles at play in the current Internet: The (Physical) Infrastructure Provider (PIP), which owns and manages an underlaying physical infrastructure (called "substrate"); the connectivity provider, which provides bit-pipes and end-to-end connectivity to end-users; and the service provider, which offers application, data and content services to end-users.

However, the distinction between these roles is often hidden inside a single company. For example, the division inside an ISP that is responsible for day-to-day operation of the network is rarely the one that is planning and specifying the evolution of the network. By identifying these players, we can on the one hand identify different business opportunities and on the other hand disentangle the technical issues from the business decisions.

When considering the kind of network virtualization that enables the concurrent existence of several, potentially service-tailored, networks, a new level of indirec-

tion and abstraction is introduced, which leads to the re-definition of existing, and addition of new, business roles:

- Physical Infrastructure Provider (PIP), which owns and manages the physical infrastructure (the substrate), and provides wholesale of raw bit and processing services (i.e., slices), which support network virtualization.
- Virtual Network Provider (VNP), which is responsible for assembling virtual resources from one or multiple PIPs into a virtual topology.
- Virtual Network Operator (VNO), which is responsible for the installation and operation of a VNet over the virtual topology provided by the VNP according to the needs of the SP, and thus realizes a tailored connectivity service.
- Service Provider (SP), which uses the virtual network to offer his service. This can be a value-added service and then the SP acts as a application service provider, or a transport service with the SP acting as a network service provider.

In principle, a single company can fill multiple roles at the same time. For example it can be PIP and VNP, or VNP and VNO, or even PIP, VNP, and VNO. However, we decided to separate the roles as it requires different groups within the company. For example, running an infrastructure is fundamentally different from negotiating contracts with PIPs about substrate slices. This in turn is fundamentally different from operating a specific network, e.g., an IP network for a service provider, which is the task of the VNO. As such, splitting the roles increases our flexibility in terms of identifying the players, the corporate enterprises, that have that role. Thereby, it keeps the economic tussles away from the technical domain.

Note that both, a PIP as well as the VNP, deliver a virtualized network. Therefore, a VNP can act as a PIP to another VNP. However, one has to keep in mind that a VNP in contrast to a pure PIP has the ability to negotiate contracts with other PIPs and to assemble networks. Moreover, there may exist multiple stacked VNP roles in reselling scenarios, or when large infrastructure owners decide to substructure into divisions and sites.

3.3 Control Plane

By default, we assume information hiding in order not to interfere with the applicability of the architecture. We outline the nature of information flowing between roles, define required interfaces, describe mechanisms for out-of-band access of migratable virtual resources, and sketch possible ways of terminal attachements in case of mobile endpoints which are not permanently attached.

3.3.1 Information flow

Communication between roles involves two major subjects: requirements and state. We assume both to be stated at arbitrary levels of abstraction. For instance, a customer may request actual infrastructure in the networking sense. He then would formulate a topology of network equipment annotated with basic resource requirements, such as 'minimum amount of RAM', HDD space, bandwidth etc. . Another customer may not care about the topology requirements, and request only host resources, providing a matrix of connection properties between his devices. Yet another customer may be interested only in service level aspects. The requirement description may then be based on resources, such as 'support for 50 concurrent user sessions of service type A'.

Finally, any requirement specification may contain a mix of elements of the above given extremes. However, we assume that requirements are refined on their way down to the hosting PIPs, and converge towards a description in traditional infrastructure terms. They are bound to converge towards a topology description (of the resulting embedding, which may be communicated upwards as state information). As we are in a networking context, which implies the existance of an interconnecting topology between requested infrastructure entities, we use the term of 'topology' in the remainder of this chapter. However, we do this without implying that this 'topology' is necessarily fully specified or is represented in a form intuitively grasped as 'topology description'.

3.3.2 Interfaces

In the following we identify the control interfaces (see Figure 3.1) in our architecture by discussing how the various players interact in order to setup a VNet. To begin with, the SP hands the VNO his requirements. Then the VNO needs to add his requirements and any constraints he imposes on the VNet. This description is subsequently provided (via *Interface 1*) to the VNP of his choice, which is in charge of assembling the VNet. The VNP may split the request among several PIPs, e.g., by using knowledge about their geographic footprints, and send parts of the overall description to the selected PIPs (via *Interface 2*). This negotiation may require multiple steps. Finally, the VNP decides which resources to use from which PIP and instructs the PIPs to set up their part, i.e., virtual nodes and virtual links, of the VNet (*Interface 3*). Now, all parts of the VNet are instantiated within each PIP but they may still have to be interconnected (*Interface 4*). The setup of virtual links between PIPs—in contrast to Interface 3—needs to be standardized in order to allow for interoperability across PIP domains. Once the whole VNet has been assembled, the VNO is given access to it (*Interface 5*). This interface is also called "Out-of-VNet" access and is necessary as, at this point in time, the virtual network itself is not yet in operation. Thus, a management interface outside of the virtual network

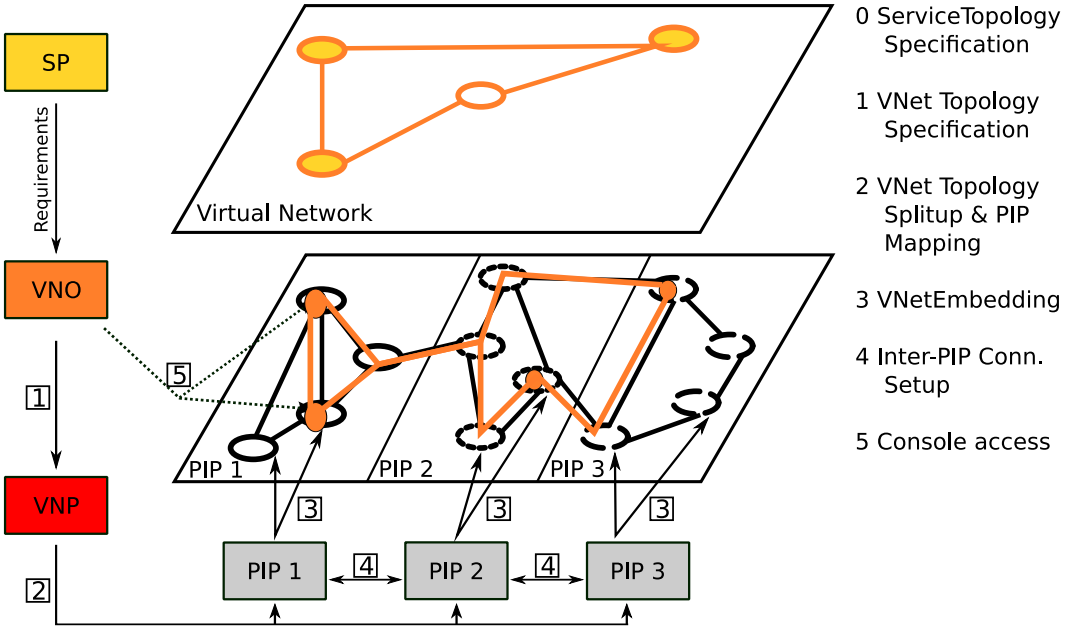
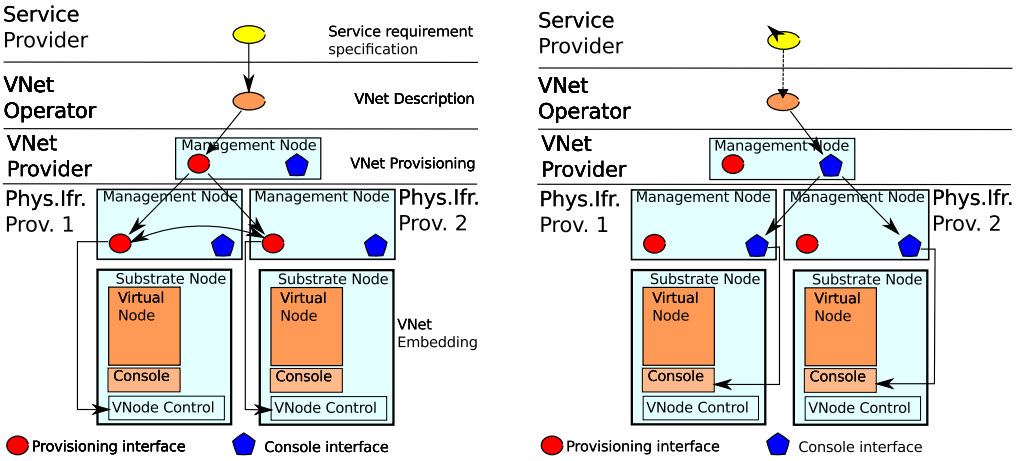


Figure 3.1: **Interfaces between players**

is needed. Once the virtual network has been fully configured by the VNO and the service is running, end-users can connect to the virtual network (*Interface 6*).

We now discuss how each player benefits from this virtualization architecture: With VNets, PIPs can better account for the constraints imposed by the customers. For example, before scheduling maintenance work or for traffic engineering purposes, they might migrate some virtual nodes to minimize downtime or to optimize their traffic flow. This is possible as long as the the new location is embedding-equivalent, i.e., satisfies all of the requirements and imposed constraints, and enabled by the level of indirection introduced by our architecture and the use of modern migration mechanisms[48, 126]. For VNPs, migration between PIPs offers a mechanism to optimize their revenue by choosing competitive and reliable PIPs. As pointed out by [141], the removal of the requirement for individual negotiations between VNOs and all participating PIPs facilitates the entry of new players into the market. Furthermore, SPs may outsource non service specific network operation tasks to other entities and thereby concentrate on their core interests relating to their respective business model. Migration processes are transparent to the VNOs. Note that they cannot trigger migration directly; however, by altering their requirements, VNOs may indirectly initiate resource migration.



(a) Provisioning (b) Console

Figure 3.2: **VNet provisioning (a) and console architecture (b).**

3.3.3 VNet Instantiation

Setting up a VNet, see Figure 3.2 (a), starts from a VNet specification. For this we need a resource description language for both the VNet topology, including link/node properties, as well as service level requirements. This description language should neither be too constrained – to allow the VNP and the PIPs freedom for optimizations – nor too vague – to enable a precise specification. We address this topic in Chapter 4.

To setup the VNet each player, for its domain, has to: formulate resource requirements, discover potential resources and partners, negotiate with this partners based on VNet resource description, and construct the topology. In the following, we give an overview of the respective tasks:

SP: The SP specifies his service specific requirements which might include a VNet topology. In addition, he may specify the kind of interface it needs for service deployment and maintenance, e.g., what type of configuration (i.e., *console*) access. He then delegates the instantiation to the VNO of its choice. Once the VNet is instantiated the SP deploys his service using the interface provided by the VNO.

VNO: The VNO uses the specification it receives from the SP and generates a VNet specification. It then negotiates with various VNPs on the basis of the VNet specification. Once a VNP is selected the VNO has to wait for the VNP to assemble the VNet. When it has access to the VNet which consists of a data and a control network it can use the control network, also referred to as the management console interface or out-of-VNet access (see Section 3.3.5), to

instantiate the network service. Finally, it may instantiate the control interface needed by the SP.

VNP: Upon receiving the VNet resource description, the VNP identifies candidate PIPs and splits the VNet resource description into multiple subsets. It then negotiates with the candidate PIPs regarding the necessary substrate resources. Once the PIPs have assembled the pieces of the VNet it completes it. Finally, it provides a management console access for the whole VNet by relying on the management interfaces of the PIPs. Note, that a VNP may aggregate requests for multiple VNets. It may also request additional resources from the PIPs to satisfy future requests. In this sense a VNP can act as any reseller would.

PIP: Based on the VNet topology descriptions a PIP receives it identifies the appropriate substrate resources and allocates them. The PIP has the ability to migrate other VNets in order to free resources for new requests. After setting up the VNet on its substrate he returns both the data and the control part of the VNet. The control part includes the PIP level management consoles to allow the configuration of the virtual nodes. Since VNets may span across multiple PIPs some virtual links may have to be setup across PIPs.

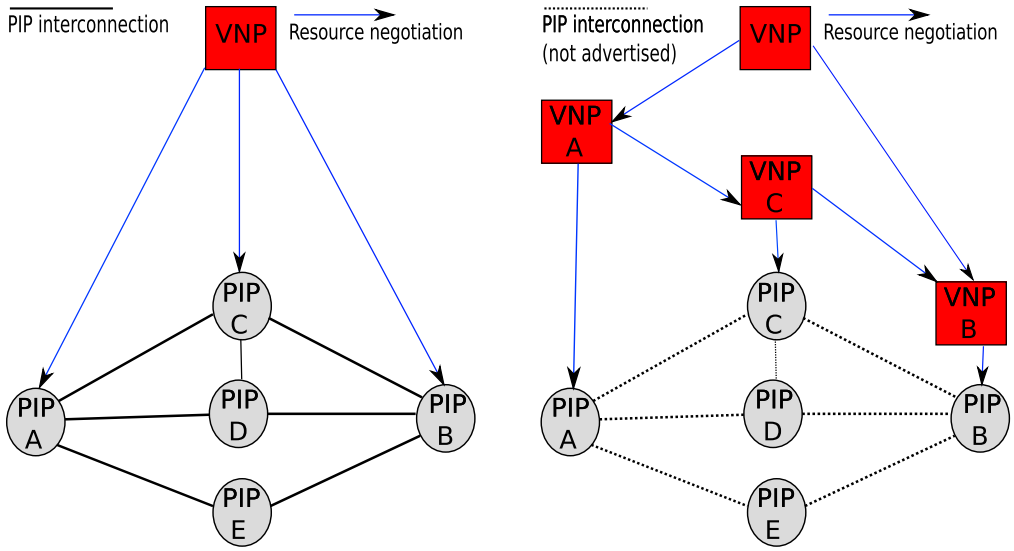
Figure 3.2 shows the resulting chain of indirections by which requests are routed by default. Each pair of interacting roles thereby acts as customer and provider, and every customer has access to its provider's provisioning and console interface. PIPs configure their resources via the substrate network element's (*SNE*) control interface, or relay console connections to the respective console interface of the SNE.

3.3.4 VNet mapping

A PIP's main task is the assignment (*mapping*) and implementation (*embedding*) of virtual resources to substrate resources. A VNP's task is arguably the same, if on a PIPs instead of substrate nodes (*SNodes*). Yet, interconnections between PIPs may or may not be known to the VNP. Depending on the scenario, he may or may not hold authoritative control over their configurations via Interface 4 depictured in Figure 3.1 (e.g., compare a broker negotiating with ISPs to a central authority handling sites within a substructured ISP).

Considering the two extremes, where either all interconnections are known to, or where all interconnections are hidden from the VNP, we see two conceptual approaches. Both of these (iterative and recursive mapping) are compatible to each other (i.e., may be intermixed in hybrid scenarios) and depictured in Figure 3.3.

Assume that the VNP decided to place one virtual hosts into each of the domains of PIP A and PIP B. The hosts are to be interconnected by a link to be commonly provided by PIP A, PIP B, and transit PIPs. In the iterative example, the VNP knows all PIP interconnections and is able to directly negotiate with every hop on the route. If he has authority over Interface 4, he may even go as far as to



(a) Iterative (b) Recursive

Figure 3.3: **Iterative (a) and recursive (b) VNet mapping on VNP level.**

specify connection and configuration parameters. If he does not, a handshake may be performed pairwise between the PIPs, communicating either directly or via the VNP. The latter communication variant may be useful to implicitely authenticate and authorize the proceedings, as the VNP will hold contractual relationships over leased resources with each PIP, which might not be the case between the PIPs.

In the recursive example, the interconnections are only known to their endpoints. Endpoint-to-Endpoint (Ep2Ep) connectivity is provided by subcontracting between the PIPs. The PIPs may spawn their own VNP roles transparent to the VNP to this end. As customer side interfaces of PIP and VNP roles are equivalent and as VNP A and PIP A belong to the same corporate entity, they may obviously be merged into one composite role internally without compatibility problems. In this scenario, the infrastructure owners reserve their rights to optimize their link routing transparently even to the VNP at the cost of additional management overhead. If the SNode's addresses are exposed in the substrate PIP A may also use reservation protocols like e.g., NSIS[37], probing for paths and reserving resources along one of them.

As previously mentioned, hybrid scenarios are conceivable: VNP may know only about PIP A's connectivity and have a special transit tariff offer from D. In this case,

he may negotiate with A and B for host resources, with A and D about resources on their interconnection, and request connectivity between D and B in his negotiations with D. D thereafter has the option of either using his own link or recursively subcontracting C's capacities.

3.3.5 Out-of-VNet Access

Each VNet consists of two planes: a data plane and a control plane. The data plane is what one commonly refers to in the context of virtual networks. But, per default, any VNet after instantiation is just an empty set of resources that have to be configured. Moreover, a control plane is necessary during VNet operation for specific VNet management tasks. Such a management access architecture is shown in Figure 3.2 (b). Every player maintains a control interface hosted on dedicated management nodes for "out-of-VNet" access to the VNet resources. This interface provides a console interface as well as control options such as virtual resource power cycling. As there are multiple levels of indirection, it also stores the location of the subsequent control interfaces. Control requests are thus relayed from player to player until they get to the appropriate virtual substrate component. In addition, to enable migration both within the PIP as well as across PIPs, consistent interface updates have to be supported.

Typical interfaces for out-of-VNet access are *consoles*, either via tty or a graphical interface. Requests for such interactive and maybe traffic intensive connections can be handled via proxy chains. Such a proxy chain can be setup by relaying requests from one player to another while instantiating appropriate connection proxies and returning their access information. To render migration seamless, connection ruptures should be hidden, e.g., by keeping the reestablishment time of the proxy chain short. Note that optimizations (e.g., by iterative referals) are admitted, but not assumed by default.

3.3.6 Terminal Attachment

End user nodes (*Terminals*) require an interface (or *Admission Network*) to authenticate and to associate themselves via attachement points (or *Access Points*) with VNets. Figure 3.4 sketches tree basic approaches: Direct local authentication (red), local authentication with roaming (yellow), and common admission network.

Direct Local Authentication: Assumption: All (to a specific terminal) available attachement points are operated by PIPs hosting parts of the respective VNet overlay. Hence, no *A4C* (Accounting, Authentication, Authorization, Auditing, Charging) traffic needs to cross administrative boundaries.

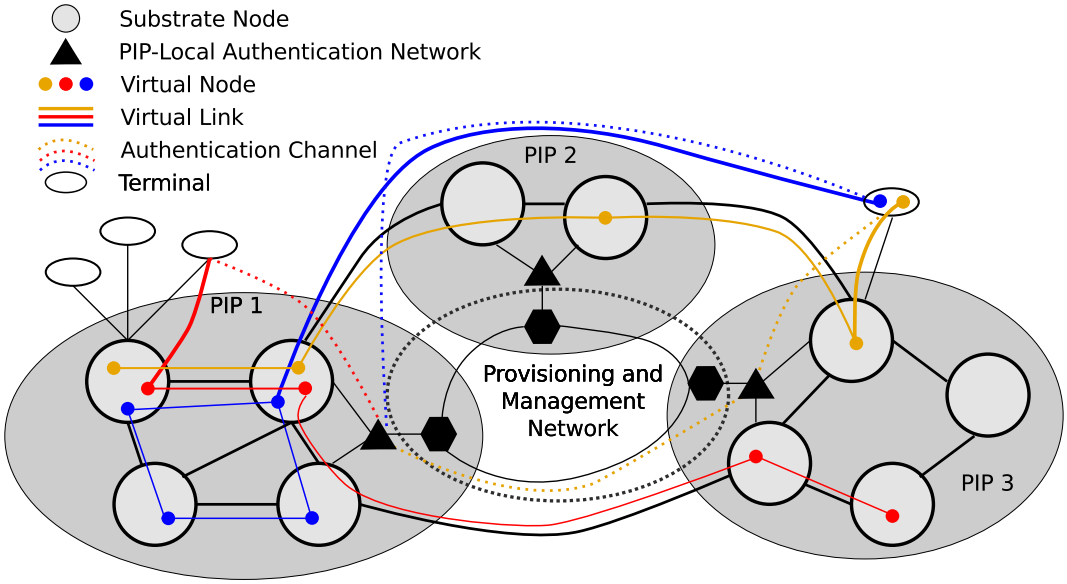


Figure 3.4: **Terminal attachment .**

Access: The terminal starts in a local admission network (e.g., VLAN), and authenticates to a local *A4CS(erver)* either directly (e.g., via Radius [101]), or a Service Portal. A successful authorization triggers a local reconfiguration, connecting the node to the requested VNet (e.g., by henceforth mapping it's traffic to an appropriate VLAN or using OpenFlow).

Advantage: End to End Quality of Service (*E2E QoS*) may be guaranteed, if the rest of the VNet supports it, as the control on network behaviour is extended to the last mile. This scenario also requires a less complex A4C Infrastructure, as no inter-PIP/VNP coordination is required for terminal authentication.

Disadvantage: The restrictions with respect to the number of access points may greatly reduce VNet attractiveness to Service Users and hence impede with strategies to introduce VNets.

Local Authentification with Roaming: Assumption: All available access points are operated by VNet-aware providers. Roaming agreements and established protocols etc. exist between these providers.

Access: The terminal requests Access in a similiar fashion to the previous scenario. However, A4C traffic may now have to cross administrative boundaries, and, depending on the nature of the roaming agreements, tunnels may have to be established, extending the VNet from the nearest access point at a PIP already hosting the requested VNet.

Advantage: E2E QoS may still be possible with appropriate roaming agreements. It is likely that concepts and frameworks of the mobile phone context may be salvageable for the VNet context in this scenario.

Disadvantage: This scenario has a higher complexity, compared to the first one, by requiring inter-operator A4C communication and tunnel establishment. This may require cooperation of PIPs not involved in the specific requested VNet.

Common Admission Network: Assumption: A common admission (V/)Net without explicit roaming agreements is provided for the terminals. This admission network may be global (e.g., the Internet), or of local scope with respect to a set of Vnet aware PIPs (dedicated admission network for specific VNet providers or collaboration of providers).

Access: The terminal first gains access to the admission network. This happens either directly/implicitely (e.g., admission network connected to WiFi access point), or explicitely (e.g., requiring dialup to an Internet network access provider), depending on the nature of the VNet. It then proceeds to contact a Service Portal of its VNet Service Provider (e.g., a Web portal, a Web Service, an IPSec [116], or OpenVPN endpoint), and subsequently establishes a tunnel to an endpoint designated by the service portal. If the local PIP is VNet aware, or even participating in the specific requested VNet, this last step may be handled identically to above described scenarios.

Advantage: A common admission network already exists in the form of the Internet. Tunnel protocols are supported even by some mobile phones via different communication channels (e.g., GSM, 3G). As this approach may be combined with the other approaches, it seems a plausible technology migration path.

Disadvantage: As connectivity properties on the last mile of the Internet are not under control of the VNO by default, E2E QoS will not be supported in the generic case.

Essentially, the first approach may be considered a simplified variant of the second, and an intermediate step between the third and the second. We therefore consider it viable to start with the third and to proceed to the others subsequently.

3.4 Related Work

Over the last years, virtual network architectures have been an area of active research. Some groups have focused on using network virtualization to enable larger-

scale and more flexible testbeds. Other groups aim at virtualizing production networks or even the Internet.

3.4.1 Architectures for testbeds

Virtualization plays a key role in creating flexible testbeds for Future Internet research.

Planetlab family:

PlanetLab [31] is a highly successful example of a distributed, large scale testbed. PlanetLab has a hierarchical model of trust which is realized by *Planet Lab Central* (PLC). PLC is operated by the PlanetLab organization and is the ultimately trusted entity that authorizes access to the resources. Other actors are the *infrastructure owners* and the *users* that run their research experiments on planet lab. For each experiment virtual machines on various nodes are grouped to *slices* that can be managed and bootstrapped together. As the deployed virtualization mechanism offer only container based virtualization capabilities at the system level and do not virtualize the network stack PlanetLab offers no network virtualization as such.

VINI, as proposed by Bavier et al. [32], is a testbed platform that extends the concept of virtualization to the network infrastructure. In VINI, routers are virtualized and interconnected by virtual links. As such, VINI allows researchers to deploy and evaluate new network architectures with real routing software, traffic loads, and network events. VINI supports simultaneous experiments with arbitrary network topologies on a shared physical infrastructure. VINI builds on the architecture and management framework introduced by PlanetLab by extending the management with interfaces to configure virtual links. The first implementation based on *User Mode Linux* [133] however offers only limited performance.

An updated VINI platform, Trellis [33], allows for higher forwarding performance. It introduces a lower level system virtualization architecture that uses container based virtualization techniques for both system and network stack virtualization. Therefore virtualization flexibility is limited to the user space. VINI provides rudimentary concepts for end-user attachments [1] using OpenVPN tunnels and a single central gateway. Obviously, this solution would not scale to virtualization applied on an Internet wide scale.

Downloadable distributions of the Planetlab control framework and VINI are available as MyPLC and MyVINI, respectively.

Emulab: Emulab [54] also is a very popular testbed platform. Its offers a sophisticated management and life-cycle processes and does not offer that much of a network architecture. Emulab offers virtual topology configuration based on ns2

configuration files and automatic bootstrapping of experiment nodes. Initially, Emulab focused on dedicated servers. Virtualization capabilities based on improved FreeBSD jails were added later.

GENI:

GENI [69] is a large-scale U.S. initiative for building a federated virtualized testbed aiming at providing a powerful virtualized testbed for experimental purposes. Here, all operations are signed off and managed by a central *Geni Clearing House*, which can thus be regarded as analogue to our VNP. As a possible growth path, GENI plans on supporting federated clearing houses, but its design has not yet been presented in detail. During phase 1 of the development both—VINI/Planetlab and Emulab—are used as GENI prototypes (*ProtoGeni*).

All testbed oriented architectures mentioned above do not consider several key factors relevant for virtualizing the (commercial) Internet: They assume a hierarchical trust model that centers on a universally trusted entity, e.g., the PLC/GENI clearinghouses. To overcome this limitation we consider competing players with individual administrative zones that have only limited trust and also have the desire to hide information, e.g., their topologies. Economic models and use cases are not critical for testbed designs but are crucial for the adoption of an Internet-wide virtualization architecture.

3.4.2 Architectures for Production Networks

CABO [63] proposes to speed up deployment of new protocols by allowing multiple concurrent virtualized networks in parallel. To this end, infrastructure providers are to manage the substrate resources while service providers are allowed to operate their own customized network inside the allocated slices. These slices are acquired by negotiations of service providers with a series of infrastructure providers.

This idea in refined by Cabernet [141] which introduces a "Connectivity Layer" between the above mentioned roles. This layer is responsible for the splicing of partial networks provided by the Infrastructure Layer and the presentation of a single network to the Service Layer. It facilitates the entry of new service providers by abstracting the negotiations with different infrastructure providers and allows for aggregation of several VNets into one set of infrastructure level resource reservations.

While this structure relates to our proposal, our approach differs as we propose to split the service provider and connectivity provider role into the three roles of VNP, VNO, and SP. These roles allow for a more granular splitting of responsibilities with respect to network provisioning, network operation, and service specific operations which may be mapped to different business entities according to various different

business models. Furthermore, we extend the framework by considering privacy issues in federate virtual network operations and business aspects.

Finally, parallel work in [93] and [61] considers the technical perspective of automating VNet provisioning. However, it does not discuss roles or constraints of multi-provider scenarios.

3.5 Summary

In this chapter, we presented an architecture for virtual network management based on consideration of economic tussles and real world actors. We chose a granular set of roles to allow for splicing and duplication rather than splitting of task domains. This facilitates their grouping according to various business models from basic resource provisioning (e.g., only PIP role) down to the provisioning of fully functional virtual infrastructures (stack of roles from PIP to VNO). Recursively stackable VNP roles allow for standalone resellers, subcontracting between infrastructure providers, as well as substructuring of large organisations. This flexibility allows for a range of economic tussles to be carried out with limited or no impact on the technical domain of the architecture.

In the same sense, we assumed a strict abstraction between the virtual and substrate domain by default. The substrate embedding is supposed to be transparent to customers and the virtual domain a black box to the provider. Provisioning is done by consideration of provided requirements. Migrations or embedding changes are triggered by either optimization efforts on behalf of a provider (PIP or VNP), or requirement changes on behalf of a customer. Requirement annotations on VNet requests are assumed to relate to measurables in the domain of providers. Adding more information flow between roles will not break the architecture, but may allow for optimizations: For example, substrate topology may be exposed to a corporate internal VNP role for large scale optimization. Or VNet component semantics may be exposed to a PIP to allow for communication about high level provisioning goals mapped to basic values (such as 'responses per minute') on behalf of the PIP, if the VNO wishes to outsource this mapping.

The architecture has been implemented prototypically in a proof of concept. The implementation details are given in Chapter 8. A resource description language in support of the given architecture is proposed in Chapter 4.

Chapter 4

Resource Description

Distributed provisioning of VNets involves communication about requirements and state. Offering VNet merely based on template topologies limits a customers flexibility. To surpass the provisioning of simple virtual network topology templates and to leverage the potential of truly customized networks, efforts to find flexible ways of describing VNets and their properties are underway (e.g., [29, 119]). Properties of VNets and network components include service level, and geographical locations as well as embedding constraints and may even include measurements. Some existing description languages use ontological approaches in particular to enable automated reasoning about VNets. This is not the main focus of our work. Rather, we focus on business scenarios, where the resource description language (*RDL*) is used to communicate requirements between customers and providers.

A main requirement is introduced by the insight that it is economically hardly viable for a single infrastructure owner to have a truly global footprint and presence in every location [132]. Federated provider environments and reselling are likely to form a part in VNet scenarios. We assume resellers and providers to lease infrastructure and resources from each other. This introduces more abstraction layers and implies that a plethora of different actors will partake in VNet provisioning and use. We expect these players to follow independent goals and be concerned about different aspects of the virtual network and its description. In order to keep a language useful for communication purposes (e.g., for VNet requests), it becomes important to allow for the *omission* or at least ignorance of details. Hence another requirement is the possibility to *leave arbitrary details un- or under-specified*.

To minimize, e.g., translation overhead in the context of mapping, the resource language should facilitate easy translation to ontologies used for reasoning. However, semantic value carried by objects in ontologies is of local interest, whereas the syntactic overhead is shared by all entities and provisioning components. We assume that providers will not only compete on a price basis, but tussle about service level abstractions. While one provider may provide bare resources, another one may offer, e.g., managed service infrastructures as part of the product. Both may offer the same network elements, but the second may allow for unconventional resources or properties (e.g., bandwidth on hosts, CPU cycles on routers, number of clients on 'Network' abstractions). The language has to allow for these tussles. It can compensate by relaxing its typification until the annotation of any property is both possible

and optional. This reduces the semantic value of ontological derived classes to simple 'type' descriptors. Moreover, it is difficult to handle frequent syntactic revisions of the language when multiple competing providers are involved. Tackling mutually exclusive variants is even more challenging, in particular from an operational point of view. We therefore advocate generic objects for communication purposes.

In this chapter, we first present a motivating example. We then analyze challenges for a resource description languages used to communicate requirements and mappings in a federated environment. We relate to these found challenges in a section introducing FleRD, the language we developed and use in the context of our prototype. We provide example topology descriptions, and discuss the use of FleRD in the context of dynamic provisioning scenarios. Next, we provide an overview over related work and summarize the chapter.

4.1 Motivating Example

As an example, consider the following motivating scenario: one customer requesting a VNet holds an interest in host resources (e.g., processing, memory, disk storage) and their geographic placement. She has a basic idea about which hosts need to communicate with which other hosts using specific communication channels. However, she is not interested in how the communication channels are established or their exact details. Another customer may care for complex connectivity constraints but not for the required communication components. He may wish to directly interconnect links with asymmetric bandwidth with a core link forming a shared medium without explicitly placing switches or adapters in-between. The first customer may even leave host details, such as architecture and word-size, open, but wish for redundant machines to not be co-located on the same underlying (physical) infrastructure components. Both resellers and providers may want to consider the omitted details as wildcards and fill out the gaps in the remaining parts of the description.

Customers usually do not hold interest in every detail. Allowing customers to underspecify requirements entails several advantages. All unspecified details form degrees of freedom for optimizations on the provider or reseller side. Depending on the request, available substrate components, geographic locations, or even virtualization mechanisms providing different virtual component types may be an option. Requirements may hence correlate with economic benefits of the providers and, thus, prices for the customers. Furthermore, translation of possibly abstract requirements into technical requirements should happen where competence to do so resides. Considering a service provider renting infrastructure, it is conceivable that the service provider does not know what technical requirements need to be specified in order to guarantee a certain service in a time interval, e.g., what CPU flags may be beneficial. Last but not least, enforcing the specification of irrelevant details may hamper the acceptance of a language and may be perceived as a nuisance.

4.2 Design Goals

A request formulation for a VNet may contain specification of elements, placement constraints, connected resources, their topological arrangement, as well as other properties (e.g., OS selection). It may also contain non-topological requirements which refer to more than a single element (e.g., requirements on binary compatibility). The set of possible component types and descriptive details may grow in the future and may be and hard to enumerate at any point in time. Especially, the property description may depend on the level of abstraction, based on the interest of the involved parties. Nonetheless, they should be describable and—as motivated by the multi-provider considerations—optional. We therefore identify the following challenges and goals:

Realism and generality: The description language must be able to describe real world topologies. For example, it is not sufficient to model the network as a graph, where each link has exactly two endpoints. Shared media and shared resources form part of real topologies on every level of abstraction. At the same time, some optimization problems in the substrate (e.g., link placement) are of graph theoretic nature. It is convenient if the topological description are nevertheless graph like to facilitate the application of graph algorithmic mapping approaches.

Extensibility: The language must be extensible with arbitrary network elements and properties to be future proof. Implementation changes and dictionary updates due to language extensions should be limited to provisioning components handling them rather than requiring every component to know about them. If economic tussles about abstractions are to be allowed, the syntactic impact of new or unconventional property associations to network elements should be limited.

Grouping and aggregation: The language must be possible to correlate properties of multiple network components (e.g., placement or binary compatibility). Moreover, it must be possible (e.g., for resellers) to aggregate resources from different network elements.

Omission and policies: It must allow to omit all details irrelevant to the describing entity. Moreover, white or black listing of properties should be supported to allow for different degrees of specificity. The missing details may be filled out during the mapping process, as they are determined. Omission must be possible both for components and their associated properties.

Mapping: A virtual component's resources may be split up upon embedding. They may be mapped to different types of substrate resources (e.g., disks onto RAM) or shared resources (e.g., NFS volumes or bandwidth for tunnel connections). In all these cases, mappings should be expressible and unambiguous. They

need to clearly identify the substrate components and resources hosting virtual ones.

Layered mapping: The mapping may happen in multiple iterations (e.g., to providers, to sites of a provider, on the actual substrate). While we may assume that mappings and embeddings are usually not communicated to business partners, we should allow providers to substructure centrally and gather management information on each mapping step. Moreover, the language should be generic in the sense that it cannot only describe one virtual and one physical network but also allows for virtualized substrates (e.g., a company providing virtual links via a leased low-latency virtual network). A resource description language used for communication therefore should support the modeling of multilayered mappings.

4.3 Flexible Resource Descriptions (FLeRD)

We unfold FleRD in three steps. First, we derive design principles from our requirements and introduce the basic descriptive objects. Second, we fill in detailed descriptions of individual object fields. Third, we outline the attribute property structure we use in the context of our prototype.

4.3.1 Approach

In the following, we describe the design approach chosen for FleRD and motivate it referencing to the different requirements and goals identified in Section 4.2. Throughout the section, we reference to descriptive objects of FleRD shown in Figure 4.1.

As mentioned and motivated above, we advocate the use of generic description elements in FleRD's context. As a consequence, our description model is centered around basic *NetworkElement* (NE) objects interconnected via *NetworkInterfaces* (NI) objects. Keeping these objects generic has the side effect that descriptions of resource aggregations, or non-standard entities (e.g., clusters or providers) is trivially supported. They may be modeled as *NetworkElements* of an appropriate type and included as topological elements. This may be used, e.g., to describe mappings in the context of a reseller.

NE properties are described as a set of *attribute-value pair* objects labeled as *Resource* and *Features*. The meaning of resources here is canonic and they may be shared amongst NEs. Features represent any type of property that is not a Resource (i.e., cannot be described in an amount of units; e.g., CPU flags, wordsizes, supported virtualization mechanisms, geographic locations). Associated *Feature* sets are interpreted as a logic clause: *Features* form predicates and sets of *Features* with

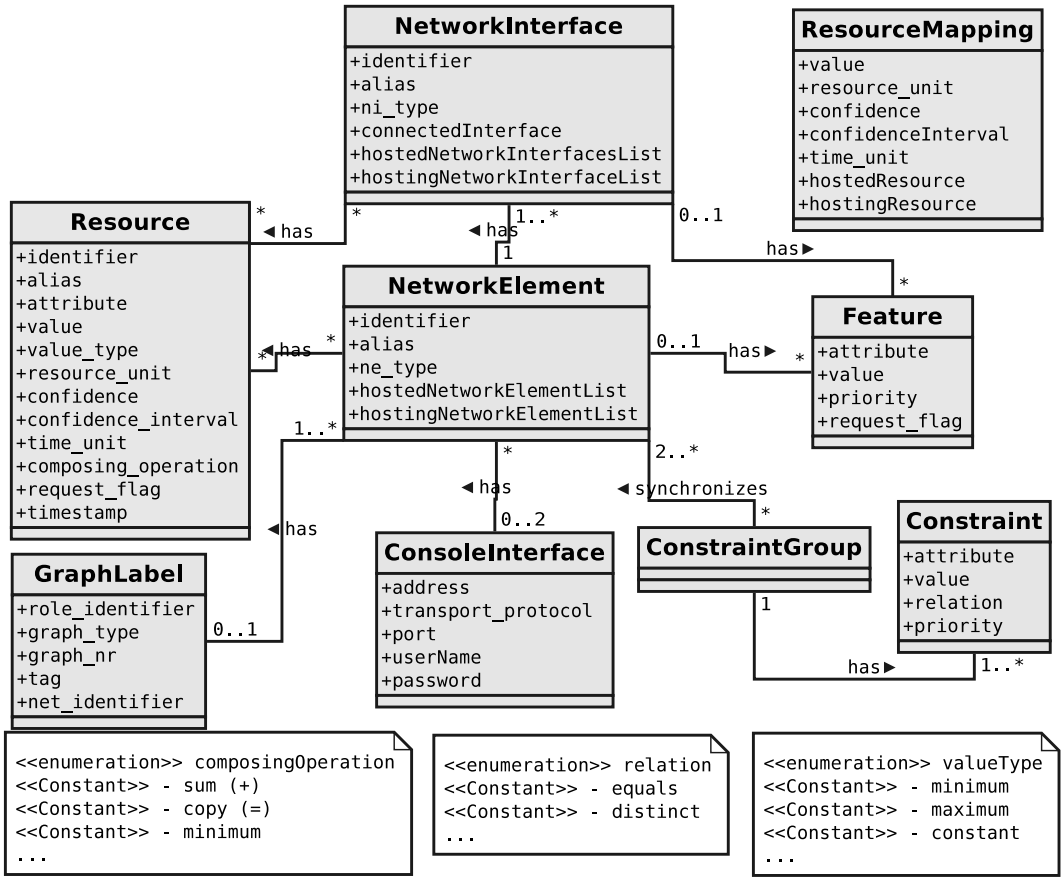


Figure 4.1: **The FleRD resource description language.**

corresponding attributes alternatives (disjunctions) within the clause. Features can also be used to explicitly state mapping choices (white listing) or used together with a corresponding attribute to state forbidden mappings (black listing).

While at first glance, the description of shared media (i.e., multi-homed links) seems contradictory to the idea of describing topologies as graphs, the concepts can be combined easily: NEs may represent both nodes and links interconnected by *NetworkInterfaces*. Interpreting NEs as vertices and interface connections as edges yields the desired result. Moreover, it becomes possible to interconnect any NE type with any other NE type in the context of the language and hence omit components practically required in embeddings.

While some requesting entity may not care about specific property values, it may still want to state correlation constraints: For example, redundant components may not be co-located (geographical or substrate position features), they need to be binary compatible (in architecture, word size, OS), or be diverse (feature value selections should ensure entropy). Also formulations like ‘At least one of the machines should

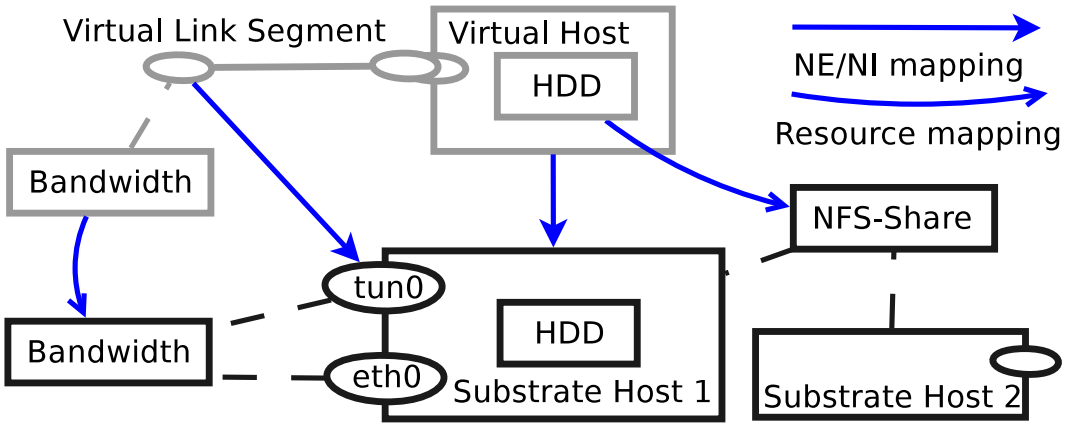


Figure 4.2: **NE/NI and resource mapping.**

be in Washington' may be relevant. FleRD allows for such formulations by means of *ConstraintGroup* objects binding a set of NEs to *Constraint* objects formulating the requirement in terms of feature or resource attributes, optional values and defined correlation functions. Again, *Constraint* sets may be considered predicates in a logic clause to be solved during the mapping steps.

It is technically feasible—and eventually may become economically viable—to introduce multiple layers of virtualization. Conceptually, the substrate does not differ from the VNet, as this is the main purpose of the abstraction of virtualization. It therefore seems natural to describe both in the same language. Allowing mappings from one topology to another of the same type allows for recursion and iterative layering, for example in the scenario where an infrastructure is sub-structured. In FleRD the *GraphLabel* object allows for identification of the different descriptions.

Mapping in FleRD follows a dual approach: NE and NI entities may be mapped directly onto other NE and NI entities, and *ResourceMapping* objects convey mapping relations between *Resources*. The rationale for this becomes clear when considering potential ambiguities in Figure 4.2: *Substrate Host 1* features both local disk space and a NFS volume to accommodate for the virtual HDD resource. Resource mapping is hence required to specify the volume used for hosting. Mapping only the virtual bandwidth resource to the substrate one on the other hand does not allow the identification of the hosting substrate interface.

Note that one of the major motivations in the design of FleRD was its adaptivity to a variety of scenarios. This implied on one hand the priorization of expressiveness over conciseness, and on the other hand the need to limit redundancy and overspecification to avoid overburdening the language. As a consequence, FleRD descriptions are essentially topology descriptions, which however may remain very coarse.

Arguing that every set of requirements that can be met relates to a topological

structure (canonically the one it is realized with), we conclude that all of these requirement sets can be expressed as such. Moreover, Feature annotations may be used to express permission to modify parts of the topology, if they are expressed merely to convey interconnection properties. Other representations may be more concise in specific scenarios (e.g., traffic matrices in scenarios where the majority of customers bears interest in complex connectivity parameters, but is not interested in topological structure at all). But they would fail to convey topological details by themselves and are redundant with respect to expressibility in combination with FleRD. Instead, they can be carried along with simplified FleRD descriptions in the context of negotiation protocols used.[1]

4.3.2 Object Fields

Resource and *ResourceMapping* objects allow for different type values (describing e.g., minima, maxima, etc.) in the description of provisioning boundaries. Heuristic specifications and mappings are possible by description of confidence levels and intervals. Units of measurement for *value* and *confidenceInterval* can be annotated as *resource_unit* and *time_unit*, respectively. The *composing_operation* field may specify a function by which mapped (partial) resources aggregate to a given value (e.g., whether mapped bandwidths correspond to sums or to the minimum of provided partial resources). FleRD offers basic support to convey measurement results as *Resource* objects using the time stamp field. However, we do not recommend its use for complex measurements for reasons outlined in Section 4.5.

Formulation of intermediate mappings may be used to express complex resource aggregations (e.g., splitting a link into parallel links and successively mapping links to paths)[2]. *GraphLabels* objects identify each layer description. In our prototype, we differentiate between virtual (*overlay OL*) network, substrate (*underlay UL*) graph and interconnecting (*mapping layer ML*) *graph_type*s. *graph_nr* and a *tag* may annotate a specific sublayer, if intermediate mappings are formulated. In the following, we sometimes refer to specific layers by their *graph_type* and *graph_nr* (e.g., OL1 or ML0). The graphs are marked by the identifier of the owner (*role_identifier*) and a *net_identifier*. It is important to note that the graph type is relative to the owner: an OL graph on behalf of an infrastructure provider may correspond to a partial ML graph of a reseller mapping parts of a VNet to different VNet providers.

NEs, resources and interfaces feature an alias field to allow for aliasing in different scopes while keeping a common identifier to which all parties can relate, unless desired otherwise and supported by suitable mechanisms. In order to allow for more granular indication of preferences in terms of unfulfillable requests, or to group

[1] e.g.: FleRD description of a single link interconnecting a large number of host devices and their interdependencies (Constraints), and a traffic matrice referencing these hosts by their identifiers

[2] This may be used, e.g., to cache steps in mapping decisions, and to save on calculation time when specific parts are changed or migrated

options together, *Features* and *Constraints* feature priority indication fields. To differentiate requests from assignments, if both are to be annotated simultaneously, the *is_request_flag* is used.

The optional *ConsoleInterface* object conveys information about how to connect to the console of the network element. Our prototype uses canonic IP addresses, transport protocol, and port information in combination with username password credentials. Extensions to support other security mechanisms are easily feasible.

4.3.3 Property Attribute Space

Resource and feature attributes can be structured along ontological concepts (e.g., `/class/subclass/.../ fieldname`) to facilitate conversion for automated reasoning. They can be standardized or follow a canonically structured hierarchy analogous to, e.g., the Simple Network Management Protocol's (SNMP) MIP structure [97]. Compound attributes specifying sets of properties in one *Feature* object (e.g., with attribute: 'host_config' and value: 'std_offer1') allow to limit the number of necessary descriptive objects. They may require standardization or multilateral agreements.

However, in the context of our prototype, we do no develop a definite attribute scheme. Rather, we use a tree-like canonically extensible attribute structure in the context of our prototype. In our case, increasing depth corresponds to increasing specificity. Feature attributes therefore usually consist of a part identifying the *NetworkElement* or *NetworkInterface* type, and a second part designating the described property (e.g., `/node/host/.../wordsize`). Resource attributes (e.g., `/link/symmetric/bandwidth`) are structured likewise. To allow for arbitrary levels of specificity, every level on the hierarchy contains a `.../generic/` subtree, featuring the union of all properties (aforementioned second part) of the respective other subtrees. In order to avoid issues with semantically equal but differently labeled leafs (yielding redundancy or collisions in the generic subtrees), we recommend standardization of the leafs and reserved subtrees for private extensions.

4.4 A Webservice Use Case

In order to give an example, in this section, we attend to a concrete use case. For the full details of the use case presented in this section, we refer the reader to the FleRD XML schema in appendix Chapter A and the example in appendix Chapter B.

We consider a VNet request (e.g., issued by a service provider) consisting of two virtual hosts *VHost1* and *VHost2* that are connected by two interfaces. For example, this VNet may describe a *webservice*, offering some service (of type *A*), and which is distributed over two processes. The VNet specifies that *VHost1* can either be located in the U.S. or in Canada (*white listing*), while the location of *VHost2* is

completely unspecified (i.e., *vague*). Moreover, the interfaces connecting the two virtual hosts forbid a connection via Afghanistan (*black listing*), and specify that the connection must be redundant in the sense that it is realized as two disjoint paths.

Prior to mapping such a request to his own infrastructure, a provider may want to refine requirements and VNet topology within the given bounds towards his own optimization goals. He also may need to translate high level requirements into low level ones. It is optional to save intermediate descriptions and keep the original request for reference. These intermediate descriptions may also be considered cached intermediate results of a more complex embedding process. In our example (and in the context of our prototype), such descriptions are labelled *Overlay*.

Likewise, the provider may use refined views on his own infrastructure as a basis for the mapping process rather than a crude view on his physical infrastructure. Again, multiple views may be stored for reference. In our example, these descriptions are labelled *Underlay*.

Finally, a *Mapping Layer* connects *Overlays* to *Underlays* and provides a view on embedding state. In our prototype, this layer also stores configuration parameters specific to virtual network elements.

In the following, we will follow the different stages of mapping a VNet request to a physical *Underlay* network. For simplicity and clarity, we assume the perspective of a single role, but the requests could also be communicated between different roles. Note that the labels of *Overlay*, *Mapping Layer*, or *Underlay* are specific to each role. Across roles, the *Overlay0* of a provider may be, in fact, part of the *OverlayN* or *Mapping Layer* of his customer.

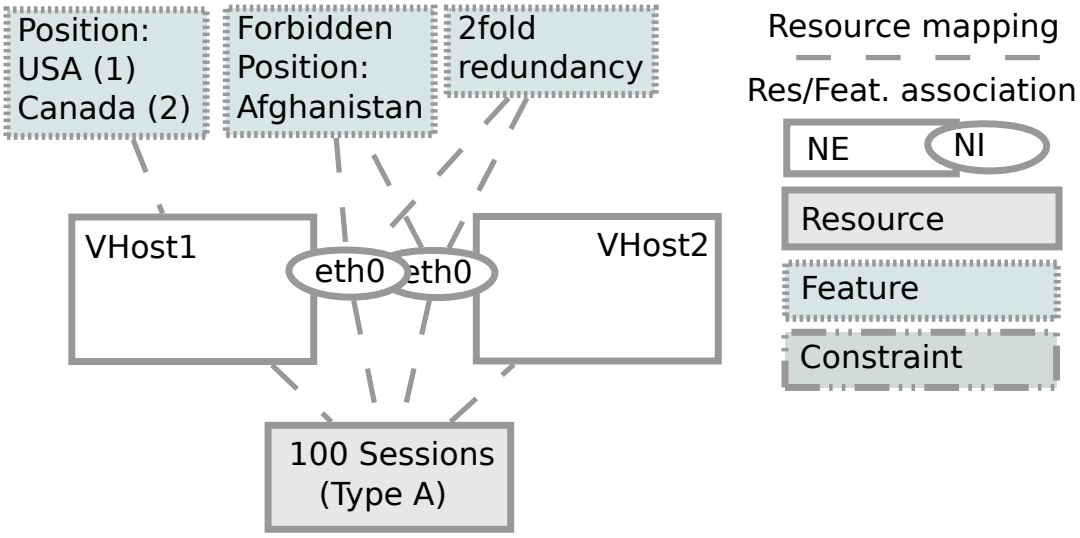


Figure 4.3: **Overlay0 (OL0).**

Figure 4.3 illustrates the Network Elements, Resources, Features and Constraints of the VNet request (the so-called *Overlay0*). To give a concrete example, in an XML-based version of our language, the white listing is described as follows:

```
<features>
  <!-- X: whitelist
    - this node may be hosted in the US (preferred) or Canada -->
  <Feature>
    <attribute>/position/continent/country</attribute>
    <value>/NA/USA</value>
    <priority>1</priority>
    <request_flag>true</request_flag>
  </Feature>
  <Feature>
    <attribute>/position/continent/country</attribute>
    <value>/NA/Canada</value>
    <priority>2</priority>
    <request_flag>true</request_flag>
  </Feature>
</features>
```

The original VNet request is subsequently expanded and complemented with concrete Resource annotations: the virtual hosts have RAM and HDD resources and the virtual links have requirements for upstream and downstream bandwidths as well as for the tolerable latency. In order to realize the requested redundant link, two multiplexing components are added to split the link into two disjoint virtual paths. A Constraint is used to ensure that the virtual links are mapped along disjoint paths, i.e., along resources with physically distinct positions. Figure 4.4 summarizes the resulting request (of *Overlay1*).

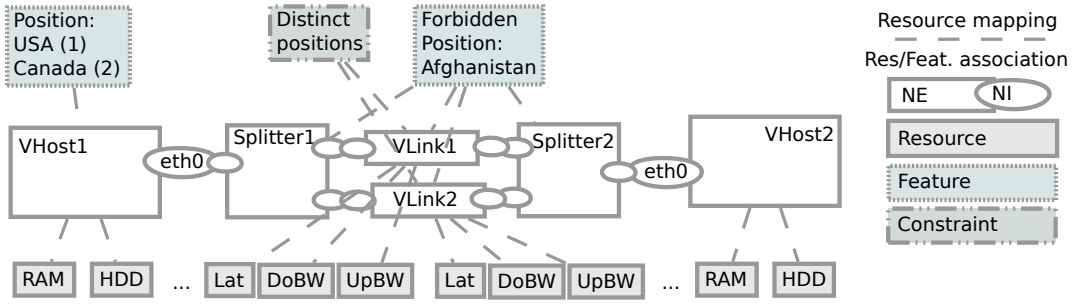


Figure 4.4: **Overlay1 (OL1).**

Both *Overlay0* and *Overlay1* represent requests, i.e., the *request_flag* is set. The remaining layers do not set this flag in our example. The *Mapping Layer* constitutes the "bridge" between request and substrate. It can contain annotations of the mapping or also store state (e.g., on which elements have been communicated to

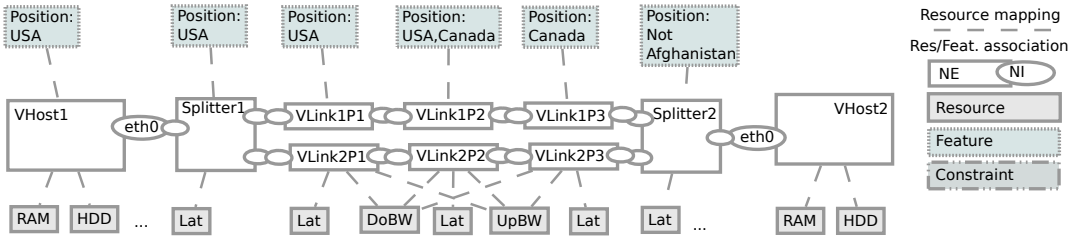


Figure 4.5: **Mapping Layer (ML0).**

which other business roles). In Figure 4.5 an example is given, where the virtual links are expanded into three virtual hops, one in the U.S., one in Canada, and one between the two countries.

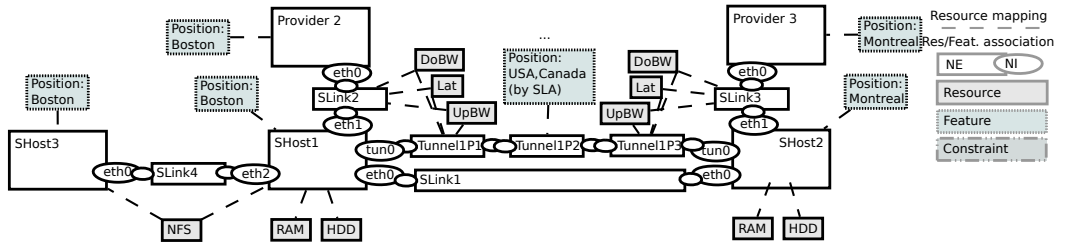


Figure 4.6: **Underlay1 (UL1).**

Finally, FleRD is used to describe the underlay. In our case, it is split into two parts, an *Underlay0* and an *Underlay1*. In Figure 4.7, *Underlay0* describes the actual physical setup, where, e.g., an NFS resource is only associated to *SHost3*. In Figure 4.6 on the other hand, *Underlay1* describes the logical setup of the substrate actually used in mapping calculations. In this setup, the NFS resource is associated to all devices it is exported to, and an existing tunnel via *Provider2* and *Provider3* is modeled.

In our example, *VHost1* is mapped to *SHost3*, *Splitter1* is mapped to *SHost1*, and *VHost2* and *Splitter2* are mapped to *VHost2*. We assume that the upper virtual link is realized via three tunnel segments, and is routed via two external physical providers, *Provider 2* and *Provider 3*. The second virtual link is realized inside the given provider as a substrate link.

4.5 Support for Operational Dynamics

FleRD facilitates the description of requests or mapping states. However, as flexibility is one of the main potential advantages of virtualization, it is likely that embedding requirements change over time. Changes may follow a schedule or be

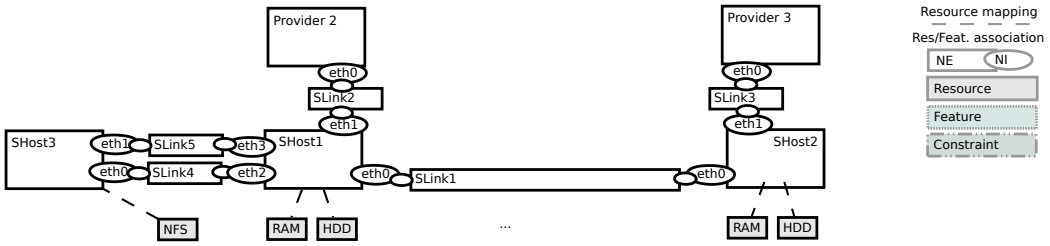


Figure 4.7: **Underlay0 (UL0).**

based on current measurement results. This raises the question of whether time annotations are required in the resource description language, and how to incorporate measurements in the description.

We believe that while it may be tempting to include time or measurement information in the description language itself, it may overburden the language. In the case of measurements a separate dedicated language can be used. This becomes apparent when considering distributed embeddings of a virtual NE: one 'measurement' may include a series of partial measurements or measurement series at different embedding locations. A single relevant parameter may depend on many different basic parameters in a complex manner. The measurement details may need to be conveyed in arbitrary formats and order. They may or may not correlate with individual NEs, network interfaces, Resources or ResourceMappings. If to be integrated in FleRD we suggest a recursively usable generic container for serialized objects which can be associated to every of the above mentioned objects.

As an unobtrusive solution for a basic measurement support, we included the optional time stamp field in the *Resource* object. This constitutes a basic means to convey actual embedding situations in comparison to requested and promised resources. Our prototype architecture assumes requirement driven communication between VNet customers and providers based on business concerns. Customers do not receive information about or access to substrate components by default. The requirements are either directly based on measurables in the provider domain, or relate to parameters the provider is able to correlate with the measurables (if provisioning comprises more than simple leasing of resources). Measurements outside one's own domain (i.e., breaking the abstraction of virtualization) is thus considered an optimization involving proprietary mechanisms between selected entities. We therefore consider it out of scope for generic languages.

The inclusion of time constraints may not only warrant extensions to all objects but imply additional side effects: Property changes or temporary removal of components may result in nontrivial topological changes, which in turn need to be expressed. On the other side, description of multiple full graphs annotated by time constraints may yield the same expressiveness — retaining unambiguity at the cost of redundant specification parts. In principle, it is possible to use the FleRD *Feature* object to

specify basic time constraints on NE granularity, and to identify coherent topology versions e.g., by corresponding tags but we do not recommend this use. Given that an annotations of time constraints on a graph level are trivially feasible by integrating them into negotiation interfaces, we advocate such a solution.

4.6 Related Work

Shared testbeds and federated prototype architectures require resource description languages to describe and communicate network specifications, services as well as embeddings. PlanetLab [31] uses the *Resource Specification* (RSpec) language to communicate network requirements between various actors. RSpec [107] is specified under the *ProtoGENI* project and is a basic resource description language comprising *nodes* and *links*. Nodes may be bound to a certain *location*, have various *interfaces*, have a *relation* with other nodes, and may host certain services. The *Network Description Language* [120] approach is based on the Resource Description Framework [103]. The original intent of this effort was to describe physical networks (in particular optical networks) to facilitate inter-domain lambda provisioning [121]. Efforts within the *NOVI* [87] and the *GEYSERS*[3] project are underway to extend the language to include the description of virtual networks. The NDL+OWL [29] language is based on an ontology paradigm which allows for automated reasoning and calculation of substrate configurations. Houidi et al. [77] propose an ontology which is also centered around the concept of a NetworkElement, but do not consider additional topics covered in this chapter. E.g., vagueness, omission, and non-topological constraints such as binary compatibility or co-location are not considered and discussed. Moreover, mapping remains restricted to two layers (virtual and physical).

The *Open Cloud Computing Interface* (OCCI) [90, 91, 102] was originally specified by the Open Grid Forum (OGF) as an API for services based on the infrastructure-as-a-service model. It has since undergone many extensions to serve services based on platform-as-a-service and system-as-a-service models. The OCCI specification consists of the *OCCI Core* [102], *OCCI Infrastructure* [90] and the *OCCI HTTP Rendering* [91] standards. The OCCI core specification and the infrastructure specification describe how to model an entity and the associated resources and links respectively, whereas the serialization of these models is demonstrated in the OCCI HTTP Rendering standards.

DEN-ng [119] is based on the DEN information model and focuses on business-driven network management solutions. VN-SLA [60] allows the formulation of Service Level Specifications (SLS) for virtual networks. Specifically, it allows to specify consequences of failure to meet specified service levels. *VxDL*[4] is a virtual private

[3] http://www.geysers.eu/
[4] http://www.ens-lyon.fr/LIP/RESO/Software/vxdl/home.html

execution infrastructure description language. VxDL describes computing and management resources while including the time limits for which these resources need to be realized. Any description in VxDL consists of four parts: general description, description of non-network resources, network topology and relevant resources, and finally the time line for which the description is valid.

These languages differ from our approach in several respects. Ontological approaches, such as NDL+OWL allow for reasoning about substrate configurations. FleRD however focusses on communication in scenarios where flexibilities required for business tussles are likely to render the associated syntactical overhead a burden. In contrast to the other languages, FleRD explicitly allows for structured multi-layer mapping descriptions on both network element and resource granularity. While VxDL and VN-SLS allow the modeling of VNets, they do not consider the formulation of mappings. DEN-ng allows for mapping, but differentiates between virtual and physical instances, thereby limiting the depth of the description. NOVI mentions the possibility of virtualized substrates, but does not consider resource mappings. Like NDL, it also allows the modeling of a (multi-homed) shared communication channel only as a full mesh of distinct links by limiting links to two endpoints. While RSpec allows for the interconnection of multiple endpoints via one shared link, resources are not modeled as separately referenceable entities. This complicates the specification of resources shared between two or more nodes. Moreover, we are not aware of any literature in the field explicitly addressing specification granularity, allowing for white and black listing of properties, or omission of both resources and components in a federated environment.

4.7 Summary

In this chapter, we analyzed the requirements for a resource description language used in the context of requirement or mapping communication. We find that specificity in requests comes at a cost in embedding flexibility, yielding an economic incentive to allow for vagueness in both request formulations and mapping assertions. Foreseeing economic tussles about abstraction and service levels, we envision an environment of frequent introductions of new virtual network element or resource types, as well as unconventional property descriptions.

Our language therefore aims at reducing of syntactic overhead while still allowing for detailed and flexible descriptions. It allows for the arbitrary omission of property as well as network component specifications, and facilitates the fine granular vagueness in terms of black- or whitelists. Both shared media and shared resources can be modeled, and mapping can be indicated on both network element and resource granularity. Partial mappings on aggregated resources (e.g., cluster) or higher abstraction levels (e.g., providers or sites) can be formulated canonically by selection of appropriate NE types. These multiple mapping steps can be cached by storing

intermediate topology descriptions, and communicated by transmission of multiple graphs identifiable by GraphLabels and related by mappings.

We design the description language to retain a graph like structure even in the presence of multi-homed links in order to facilitate the application of graph algorithmic mapping approaches, such as discussed in Chapter 5. As descriptions tend to increase rapidly in size with complex or large scenarios, we created an XML schema to facilitate the use of consistency-checking XML editors. Moreover, at the time of writing, we are working on tool support with a graphical user interface for VNet specifications and the display of state and mappings. The language is implemented and used in our prototype described in Chapter 8.

Chapter 5

Virtual Network Mapping

VNets decouple networks from the physical constraints of the underlying infrastructure (substrate). They can offer opportunities for customized network environments and can be flexibly embedded at optimal (e.g., economical) locations and even migrated. One of the central challenges arising from VNets concerns the strategies to leverage this resource allocation flexibility.

To this purpose, virtual resources need to be assigned (*mapped*) to infrastructure resources, on which they are thus instantiated (*embedded*) and thus hosted. The mapping problem has been considered in various contexts (e.g., [44, 70, 72, 140]). Previous work focuses on specific subproblems, e.g., restricts the type of networks to graphs (e.g., without shared resources or multi-homed links) or calculates approximations. We devise a generic formalization applicable to real world networks and explicitly take into consideration migration and migration costs.

As described in Chapter 4, we consider a scenario where participating entities focus on abstractions relevant to them and optimize towards their own goals. A customer requests a network with (possibly incomplete) requirement specifications. Every underspecified parameter is a degree of freedom to the providers. Brokers and resellers (e.g., VNPs) without holistic substrate topology knowledge split up the request and negotiate for partial resources with VNet providers, optimizing to their own benefits. VNet providers in turn optimize the embeddings inside their own substrate segment along their specific substrate management policies. We further assume that large VNet providers may substructure into sites, and introduce their own internal reseller or broker instances.

Mapping approaches therefore need flexibility for multiple levels of abstractions and a large number of potential optimization goals. Moreover, the type and arrival time of VNet requests is hard to predict, making it impossible to ensure efficient resource usage by static placements alone. Dynamic reconfigurations, such as resource reallocations (migrations), on the other hand come at a cost. It is crucial to identify these and to weight them against benefits of more efficient embeddings. While PIPs (see Chapter 3) map VNets to a substrate topology, VNPs arguably perform a similar task, mapping VNets onto a topology of interconnected PIPs. Our MIP is thus applicable to multiple levels of our VNet architecture.

This chapter addresses the question of how to calculate optimal embeddings while taking into account the cost constraints of migration. We formalize the problem in form of a mixed integer program (MIP) which can be applied on both in the context of VNP and PIP roles.

The remainder of the chapter is structured as follows: We first analyze challenges, requirements and migration cost factors. After introducing the key concepts, on which our MIP is based, we continue with an in-depth description of our solution. We follow with a discussion of possible extensions. We evaluate the performance of our solution on a basis of three characteristic use cases we identify, and conclude in a summary.

5.1 Challenges and Requirements

As mentioned above, we consider a scenario with multiple entites, in which providers may exploit underspecified characteristics of requested VNets for embedding flexibility. This scenario entails two important implications: First, the abstraction characteristic of virtualization and the given requirement specification allow providers on any level (VNet providers and resellers) to freely optimize embeddings for their own purposes. We assume that requirements either consist of parameters measurable in the provider's domain, or that the provider knows how to translate them[1] into measurables of his domain[2]. As long as these requirements are met, he may assume not to disrupt applications inside the VNet. Second, resource mapping—a computationally hard problem—is likely to happen in multiple steps. Resellers and brokers (i.e., VNP roles) may map resources to providers, which in turn may decide to substructure into areas or sites (i.e., VNP and PIP roles). Mapping approaches thus need to scale to the problem sizes of the respective role/player rather than infinitely.

The embedding strategies and objectives are likely to vary along with size and heterogeneity of the available substrate and the time available to find a solution. It is unlikely that one VNet embedding approach will fit all situations: Symmetries in data centers may allow for reasonable approximations, e.g., by greedy virtual network placements or aggregation of similar resources. A backbone router site may be too diverse for such an approach, but allow for the computation of optimal solutions due to its limited size. In scenarios with heavy-tailed VNet durations, short term placements may not even warrant the effort of optimization, but it may be beneficial for a provider to optimize long living VNets into an efficiently managed segment of the network. A mapping approach should hence facilitate an exchange of optimization goals and the application of approximations while allowing for the

[1] e.g., because he guaranteed abstract QoS rather than basic resources
[2] including the possibility of measurement delegation, if he he hired subcontractors

calculation of optimal solutions. It should also allow for the formulation of mapping preferences in the presence of embedding policies.

As pointed out in Chapter 4, shared media (i.e., multi-homed links) and shared resources form integral part of reality on every layer of abstraction: They exist in substrates as, e.g., wifi links and tunnels sharing bandwidth with the underlaying physical links. They may be requested for virtual components, if, e.g., the customer specifies a multi-homed virtual link that allows any two nodes to communicate with a specific bandwidth, but not necessarily simultaneously so. An ignorance of the shared resource character in such a situation is likely to overallocate resources in the substrate's backbone. Moreover, virtualization may allow for embedding of resources across resource types: Non-persistent virtual hard disks may be embedded in RAM to accelerate access speed, or vice-verca. In the case of full-duplex or even asymmetric link resources, directions of allowable flows influence the mapping of virtual resources onto substrate resources. Moreover, embedding of entire virtual networks on a single host are not only feasible, but also common to find. The mapping approach must hence allow for correct formulations of such problems.

VNet request arrive over time and an optimal placement at some time t_0 may become suboptimal at some time $t > t_0$. An algorithm to compute mere static embeddings of a given set of VNets is therefore not satisfactory in practice. For instance, if another virtual network expires, resources are freed up and another VNet could be migrated there. Re-embeddings may also make sense for network management and maintenance, e.g., to move the traffic to different paths to upgrade the routers. Moreover, the demand for a certain service can be *dynamic*, due to daytime reasons, and also the origins of the request can change due to *user mobility* or *time-zone effects*. In this case, VNets should be dynamically scaled up or down depending on the demand, and moved with the users. The arrival of a new VNet should cause minimal changes to the existing embedding of prior VNets. Since VNet migration cost is non-zero, there is a trade-off between migration cost and a superior embedding. An embedding algorithm must hence support cost-aware migrations in the sense that it trades off migration cost with the potential benefits.

5.2 Migration Cost

While migrations may yield more efficient embeddings, their costs depend on many factors. We group these costs according to three parts of the process: Resource removal, resource transferral, and resource reinstantiation.

Resource removal: A migration infers a management overhead C_{mgmt}. If, e.g., a VNP provider triggers a cross-provider migration, the termination of provisioning contracts may entail penalties C_{contract}. Temporary redundant allocations of resources and reconfiguration based service outages during migration entail

opportunistic costs C_{reconfig}. An example cause for the latter would be outages triggered by switchovers to and from transitional provisioning solutions. These may lead to reduced benefits via reduced payments from the customer whose service level agreement has been violated. All of these costs are independent of the destination of the migration. They either relate to the removal of old allocations, or to the fact that a migration is happening per se.

Resource transferral: These costs may depend on the migration destination. They relate to the actual transfer and possible property changes. Bulk data transfers may entail both real and opportunistic transit costs C_{transit}, e.g., transfer of host state may require additional bandwidth to be leased from transit providers. It may interfere with provisioning of other VNets, if routed via the same substrate links. Furthermore, adaptation may impose overhead if crucial properties change (e.g., migrating a virtual host from Xen to KVM). We denote this cost factor by $C_{\text{adaptation}}$.

Resource reinstantiation: New provisioning contracts may be required to place the resource at its new location. While this new placement is likely to entail additional costs and benefit changes, these are semantically equivalent to those considered upon initial placements. This becomes clear considering that some resources (e.g., virtual links) may be migrated by reinstantiation rather than actual transfer. We therefore do not model them separately from placement, but rather cover these implicitly in the context of placement preferences.

The mentioned costs may be zero or approximated by a bound for practical reasons. Consider the following two scenarios: The first comprises a live migration of a host inside of a (fully switched, homogeneous) rack belonging to the same provider, which provides separate links for the data plane and migrations. Evidently, contract penalties do not apply, no adaptations are required, and transit costs can be considered as destination independent. In the second, a VNP live-migrates a host between providers. All cost factors apply, but those of of resource transferral can be included in C_{reconfig}, if the migration is provided as a service on behalf of the PIPs.

5.3 Approach and Key Concepts

Among our basic assumptions is that the substrate is in itself consistent. We assume that components connected to different technologies are either capable of functioning as an converter, or are split into two separate entities that may share common resources (see Chapter 4). Thus, no incompatible technologies are directly connected in a way to create an invalid path in the substrates topology graph.

The problem can be considered a graph embedding problem constrained by basic suitability of substrate components for virtual resources. We therefore decided to formalize the problem as a Mixed Integer Program (MIP). A MIP consists of a

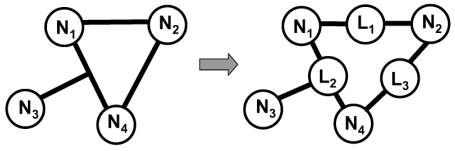


Figure 5.1: **Transform: network to graph representation.**

(linear) *objective function* expressed with a set of variables, plus a set of (linear) constraints on these variables that ensure "valid" solutions of guaranteed quality. Such MIPs can be solved by standard and highly optimized tools such as `CPLEX`. These usually provide a set of built-in approximations, some of which we consider in Chapter 6. Based on the MIP's output, a valid substrate configuration can be found.

An advantage of the mathematical programming approach is that it enables us to propose different objective functions which can be easily exchanged. For example, at some point a provider may choose to place the virtual networks on the "edge" of the physical network in order to avoid blocking bottleneck links and hence to maximize the likelihood that future VNet requests can be accepted. At another point the provider wants to spread the VNet embeddings as much as possible in order to retain available node resources at all locations. The provider may want to concentrate the VNets on as few resources as possible in order to be able to switch off other parts of the network in order to safe energy or for maintenance work.

This section introduces the basic concepts and MIP elements displayed in Tables 5.1 (sets), 5.2 (constants) and 5.3 (variables) of our mapping problem formalization.

5.3.1 Graph representation

Shared communication channels, i.e., links with several end points (both in the virtual and the substrate network) constitute a first challenge for a generic mapping approach. To describe virtual and substrate networks as classic graphs $G = (V, E)$ consisting of vertices V that are connected pairwise by edges E, we use the concept of *network elements* (NEs) introduced in Chapter 4: Network elements represent both *nodes* (set NE_{N}) and *links* (set NE_{L}). Network elements are connected by interfaces, which form the edges of the graph.

We distinguish between virtual network elements of the VNet (set $NE_{\mathrm{V}} = NE_{\mathrm{VN}} \cup NE_{\mathrm{VL}}$ of virtual nodes and virtual links) and substrate network elements of the substrate network (set $NE_{\mathrm{S}} = NE_{\mathrm{SN}} \cup NE_{\mathrm{SL}}$). In principle, any virtual node can or cannot be mapped onto specific substrate nodes, depending on the requirements. A virtual link can be embedded onto a substrate node, a substrate link, or onto a set of paths in the substrate network (resulting in a multi-flow embedding).

An example is shown in Figure 5.1. Here, $NE_{\mathrm{SN}} = \{N_1, N_2, N_3, N_4\}$ and $NE_{\mathrm{SL}} = \{L_1, L_2, L_3\}$. The purpose of the embedding algorithm is to find a mapping of the virtual networks and their elements to the network elements of the substrate. To handle links with several endpoints, we replace each link with a vertex and add graph edges accordingly.

5.3.2 Placement Policies and Suitability

We use the binary matrix $new(u, v)$ to denote whether a virtual network element $u \in NE_{\mathrm{V}}$ is mapped to a substrate network element $v \in NE_{\mathrm{S}}$ ($new(u, v) = 1$) or not ($new(u, v) = 0$).

A substrate element allocates resources for all virtual elements it hosts. To describe these allocations, we introduce the variables $alloc_{r_{\mathrm{V}}}(u, v)$ which capture the amount of virtual resource r_{V} of u hosted on v and $alloc_{r_{\mathrm{S}}}(u, v, r_{\mathrm{V}})$ describing the substrate resources r_{S} used to allocate it. The resources r_{V} requested for u are represented by the constant matrix $req(u, r_{\mathrm{V}}, s)$, where $s \in VT$ refers to the value type of request (e.g., minimum, maximum, ...). To ensure that the sum of the allocated resources never exceeds the capacities of the substrate we use the constant capacity matrices $cap_{r_{\mathrm{S}}}(v)$, $cap_{r_{\mathrm{S}}}(v, w)$, and $cap(r_{\mathrm{S}})$. The first two hold individual capacities of substrate components v and substrate interfaces interconnecting v and w with respect to r_{S}. The last represents the capacity of the resource r_{S}, itself. All three are required to correctly model various possible shared resources assignments in the substrate.

It is not always possible to map a virtual network element to some specific substrate elements. For example, a virtual node may be restricted to be placed only within the US. The constant binary matrix $suit(u, v)$ specifies whether v is suitable to host network element u ($suit(u, v) = 1$) or not.

Our mathematical program considers placement restrictions: A provider may want to bias or fix a mapping for a specific VNet according to internal placement policies or cost factors. We thus use a constant weight matrix $weight(u, v)$ to introduce a cost for each node placement. These weights can also be used as policies to prioritize certain resource allocations over others via the objective function.

5.3.3 Link Types and Resources

Next, we discuss how we handle the different link types: If the bandwidth in both directions is the same we call a link *symmetric*, otherwise it is called *asymmetric*. A *full-duplex* link supports traffic in both directions *independently*. A full-duplex link can be regarded as two independent *unidirectional* links. A shared (wireless, or non-switched, hub-like) channel is referred as *half-duplex* link. Note that half-duplex links are symmetric by nature.

We explicitly distinguish between two classes of resources $R = R_V \cup R_S$, namely virtual resources $r_V \in R_V$ and substrate resources $r_S \in R_S$. To handle the different link types, virtual half-duplex links are (e.g.—see Chapter 4) associated to an r_V of attribute `'/link/symmetric/bandwidth'` whereas substrate full-duplex links receive two r_S with `'/link/upstream/bandwidth'` and `'/link/downstream/bandwidth'`, respectively. In our embedding program, we assume a proportional relationship between r_V and r_S, that is, we consider a proportional factor $prop(r_V, r_S)$. As different functions are possible (e.g., involving constant instantiation overhead), the respective constraints of the MIP are examples.

Interestingly, differentiating between r_V and r_S in both VNet specification and MIP is not only useful for handling different link types but also for mapping nodes: It enables us to map and even split resources (see Chapter 4) of arbitrary resource types onto arbitrary other resource types. In this way, one can, e.g., map RAM requests to SWAP space or volatile low-latency HD resources to RAM.

To handle shared communication channels we decompose its multiple endpoints into a *set of flows*. In particular, for each link u, we introduce a set $Fl(u)$ that describes the set of possible source-sink pairs for u.Each flow $f \in Fl(u)$ inherits the requirements of u. Analogue to the *alloc* matrices, $flow_{r_V}(f, v, w)$ and $flow_{r_S}(f, v, w, r_V)$ reflect tentative resource allocations on substrate interfaces, and $new(f, v)$ denotes corresponding tentative flow mappings. Resources of these flows $f \in Fl(u)$ form the set $R_f \subset R_V$.

In its current version, our MIP does not consider interface bottlenecks (as opposed to link bottlenecks), and it assumes only one interface coupling between two given network elements (no parallel edges). Possible extensions to this respect are outlined in Subsection 5.5.2.

5.3.4 The Flow Problem

While we consider virtual nodes to be atomic in the context of our MIP, virtual links can be realized either as single path or multiple paths within the substrate network. The aggregated resources of the paths must satisfy the requirements of the virtual link while not exceeding the capacity limits of the substrate elements. For instance, the sum of the bandwidths of the different paths must equal or greater than the link's bandwidth demand. This constitutes a *flow problem*. However, since we tackle placement and embedding at the same time this corresponds to a multi-commodity flow problem with a twist: The endpoints are not fixed, and candidate locations overlap.

Our mathematical program ensures that the allocated flows are connected and consistent with the requirements and capacities. We enforce a *flow preservation invariant*, that is, we guarantee that the amount of flow arriving at a node equals the amount of flow leaving the node. However, we must exempt the source and the sink

of the flow from this invariant: We ensure that the traffic leaving the source equals the demand of the virtual link. The link's sink simply consumes the incoming flows. This is implemented via selector variables that render the constraint trivially true for endpoints (a tautology).

Latencies are not considered in the current version of the MIP. Respective extensions are outlined in Subsection 5.5.5.

5.3.5 Migration Support

As mentioned, VNet requests typically arrive over time and the provider faces the problem of how to embed a new VNet given the existing allocations of other requests. Clearly, a complete re-embedding of all requests is out-of-question, as this potentially comes at a high cost and with long outage times. However, small local reconfigurations may reduce the overall resource overhead and improve the overall embedding substantially or even enable the embedding to begin with.

To this end, we introduce matrices and constraints that allow the specification of reconfiguration costs and enable the solver to weight them against the respective benefits. Analogue to $new(u, v)$, we use the constant binary matrix $old(u, v)$ to describe existing mappings, and specify whether a virtual network element u is currently mapped to a substrate element v ($old(u, v) = 1$) or not ($old(u, v) = 0$).

We account for cost of migration in two respects: Destination independent costs are reflected in the constant penalty matrix $penalty(u) = C_{\text{contract}}(u) + C_{\text{mgmt}}(u) + C_{\text{reconfig}}(u)$. Destination dependent C_{transit} and $C_{\text{adaptation}}$ cost factors to migrate virtual network element u from its current position to substrate element v are summed up in the constant matrix $transit(u, v)$.

Node migration is typically more expensive relative to link migration, as links do not involve state or bulk data transfers but are rather re-instantiated. As long as at least one end point of a modified link remains in place (i.e., the connected virtual node did not migrate), connectivity can be guaranteed by temporary redundant resource allocations. Costs can be reflected by the link's $transit(u, v)$ variables. If, however, all endpoints are migrated simultaneously, at least one interconnecting segment or tunnel is required to allow for live-migrations. As a first-step simplification, we consider a scenario where every migrating host receives a temporary tunnel, as proposed in VROOM [126], and where costs are added to the respective host's $transit(u, v)$ values. Furthermore, we do not consider contract penalties for removing link segments or objective functions maximising migrations in this step. We thus assume $penalty(u) = \epsilon$ for $\forall u \in NE_{\text{VL}}$ and an arbitrarily small $\epsilon > 0$, unless stated otherwise. Section 5.5 sketches extensions to remove these constraints.

5.4 The Embedding Program

Based on the above ideas we now describe the MIP in details, see Tables 5.1(sets), 5.2(constants), 5.3(variables) for a summary. While we introduced most sets, variables, and constants above we now describe the remaining ones which support specific objective functions. We explain the constraints.

Sets	
NE_{V}	Virtual Network Elements
NE_{VN}	Virtual Nodes
NE_{VL}	Virtual Links
NE_{S}	Substrate Network Elements
NE_{SN}	Substrate Nodes
NE_{SL}	Substrate Links
R_{V}	Set of Virtual Resource
R_{S}	Set of Substrate Resource
$R_f : R_f \subset R_{\mathrm{V}}$	Set of Virtual Flow Resources
VT	Value Types
$Fl(u)$	Flows ((source,sink)-Tuples)

Table 5.1: **Set definitions**

Table 2: Constants		Range	
$weight(u,v)$	Resource Weight	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	$\in [0,1]$
$penalty(u)$	Migration Cost	$\forall u \in NE_{\mathrm{V}}$	> 0
$transit(u,v)$	Costs transferring u resources to v	$transit(u,v)$, $\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	≥ 0
$old(u,v)$	Old Mapping	$\forall u \in NE_{\mathrm{V}}$, $v \in NE_{\mathrm{S}}$	$\in \{0,1\}$
$suit(u,v)$	Suitable Mapping	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	$\in \{0,1\}$
$cap_{r_{\mathrm{S}}}(v)$	Capacity of v w.r.t. r_{S}	$\forall v \in NE_S, r_S \in R_S$	≥ 0
$cap_{r_{\mathrm{S}}}(v,w)$	Connection Capacity	$\forall (v,w) \in NE_{\mathrm{S}}^2, r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$cap(r_{\mathrm{S}})$	Resource r_{S} Capacity	$\forall r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$req(u, r_{\mathrm{V}}, s)$	Resource Request	$\forall u \in NE_{\mathrm{V}}, r_{\mathrm{V}} \in R_{\mathrm{V}}$, $s \in VT$	≥ 0
$prop(r_{\mathrm{V}}, r_{\mathrm{S}})$	Scaling Factor	$\forall r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$weight_{r_{\mathrm{S}}}$	Load Weight Factor	$\forall r_{\mathrm{S}} \in R_{\mathrm{S}}$	$\in [0,1]$
c	`sum`,`max` Load Priority Factor		$\geq \sum_{r_{\mathrm{S}} \in R_{\mathrm{S}}} weight_{r_{\mathrm{S}}}$
$min\_alloc_{r_{\mathrm{V}}}$	Min. r_{V} allocation unit	$\forall r_{\mathrm{V}} \in R_{\mathrm{V}}$	≥ 0

Table 5.2: **Constant definitions**

5.4.1 Objective Function

What is an optimal mapping for a VNet over a set of resources in a physical substrate network? The answer depends on the goals of the mapping entity and also relies crucially on the predictability of future resource requests. Even with good predictions, an optimal solution found at time t_0 may be suboptimal upon the arrival of the next request at some time $t > t_0$.

The strict scenario dependency and the number of antagonistic goals make it impossible to cover all objective functions explicitely in the context of this work. We

Table 3: Variables		Range	
$alloc_{r_S}(u, v, r_V)$	Allocated Resources	$\forall u \in NE_V, v \in NE_S, \forall r_V \in R_v, r_S \in R_S$	≥ 0
$alloc_{r_V}(u, v)$	Hosted Resources	$\forall u \in NE_V, v \in NE_S, \forall r_V \in R_V$	≥ 0
$new(u, v)$	Mapping Matrix for Elements	$\forall u \in NE_V, v \in NE_S$	$\in \{0, 1\}$
$new(f, v)$	Mapping Matrix for Flows	$\forall f \in Fl(u), v \in NE_S, \forall u \in NE_{VL}$	$\in \{0, 1\}$
$mig(u)$	Migration Selector	$\forall u \in NE_V$	$\in \{0, 1\}$
$flow_{r_S}(f, v, w, r_V)$	Allocated Resources for Flow	$\forall (v, w) \in NE_S^2$, $\forall f \in Fl(u), r_V \in R_V, r_S \in R_S, \forall u \in NE_{VL}$	≥ 0
$flow_{r_V}(f, v, w)$	Hosted Resources for Flow	$\forall (v, w) \in NE_S^2$, $\forall f \in Fl(u), r_V \in R_V, u \in NE_{VL}$	≥ 0
$load(r_S)$	Load on Resource r_S	$\forall r_S \in R_S$	≥ 0
max_load	Max Load over All r_S		≥ 0

Table 5.3: **Variable definitions**

therefore consider only two exemplary antagonistic objective functions: Minimizing resource usage (localizing embeddings) and load balancing (spreading embeddings). However, one advantage of our mathematical programming approach is the ease in exchanging different objective functions with only limited implementation effort.

An example objective that may make sense for infrastructure providers is to minimize the amount of substrate resources used for allocations. Such an objective function aims at compact VNet placements, as virtual link resource allocations can be assumed to have the highest impact here (assuming virtual hosts to require the same node resources at any location). Other motivations may be, e.g., to be able to switch off alternative components to save energy. The objective function we use for most of our experiments therefore balances resource usage against migration cost:

$$\sum_{u \in NE_V} \sum_{v \in NE_S} \sum_{r_S \in R_S} weight(u, v) \cdot alloc_{r_S}(u, v, r_V)$$
$$+ \sum_{u \in NE_V} \left(penalty(u) \cdot mig(u) + \sum_{v \in NE_S} transit(u, v) \cdot new(u, v) \right)$$

In our prototype, we also use an objective function that seeks to distribute the load equally among all network elements to minimize peak loads and congestion. Such an objective function may make sense, if requests are likely to involve placement constraints, or if resource guarantees allow for usage spikes:

$$c \cdot max\_load + \sum_{r_S \in R_S} load(r_S)$$
$$+ \sum_{u \in NE_V} \left(penalty(u) \cdot mig(u) + \sum_{v \in NE_S} transit(u, v) \cdot new(u, v) \right)$$

To this end, we extend the program by the $load(r_S)$ matrix capturing the individual substrate resource loads (needed in the objective function for efficient allocation).

Nodes:

map_node:	$\sum_{v \in NE_S} new(u,v) = 1$	$\forall u \in NE_{VN}$
set_new:	$alloc_{r_S}(u,v,r_V) \leq cap_{r_S}(v) new(u,v)$	$\forall u \in NE_{VN}, v \in NE_S, r_V \in R_V, r_S \in R_S$
req_min:	$alloc_{r_V}(u,v) \geq new(u,v) req(u,r_V,s)$	$\forall u \in NE_{VN}, r_V \in R_V, r_S \in R_S, s = \texttt{minimum}$
req_max:	$alloc_{r_V}(u,v) \leq new(u,v) req(u,r_V,s)$	$\forall u \in NE_{VN}, r_V \in R_V, r_S \in R_S, s = \texttt{maximum}$
req_con:	$alloc_{r_V}(u,v) = new(u,v) req(u,r_V,s)$	$\forall u \in NE_{VN}, r_V \in R_V, r_S \in R_S, s = \texttt{constant}$

Mapping:

relate_V:	$alloc_{r_V}(u,v) \geq min\_alloc_{r_V} \cdot new(u,v)$	$\forall u \in NE_V, v \in NE_S, r_V \in R_V$
allowed:	$suit(u,v) \geq new(u,v)$	$\forall u \in NE_V, v \in NE_S$
ne_capacity:	$\sum_{u \in NE_V} \sum_{r_V \in R_V} alloc_{r_S}(u,v,r_V) \leq cap_{r_S}(v)$	$\forall v \in NE_S, r_S \in R_S$
capacity:	$\sum_{v \in NE_S} \sum_{u \in NE_V} \sum_{r_V \in R_V} alloc_{r_S}(u,v,r_V) \leq cap(r_S)$	$\forall r_S \in R_S$
load:	$weight_{r_S}/cap(r_S) \cdot$ $\sum_{v \in NE_S} \sum_{u \in NE_V} \sum_{r_V \in R_V} alloc_{r_S}(u,v,r_V) \leq load(r_S)$	$\forall r_S \in R_S$
max_load:	$load(r_S) \leq max\_load$	$\forall r_S \in R_S$

Resource-Variable Relation:

resource:	$\sum_{r_S \in R_S} prop(r_V, r_S) alloc_{r_S}(u,v,r_V) = alloc_{r_V}(u,v)$	$\forall u \in NE_V, v \in NE_S, r_V \in R_V$
flow_res:	$\sum_{r_S \in R_S} prop(r_V, r_S) flow_{r_S}(f,v,w,r_V) = flow_{r_V}(f,v,w)$	$\forall f \in Fl(u), (v,w) \in NE_S^2, r_V \in R_f,$ $\forall u \in NE_{VL}$

Links:

map_link:	$\sum_{v \in NE_S} new(u,v) \geq 1$	$\forall u \in NE_{VL}$
map_src:	$new(u,v) \geq new(q_f, v)$	$\forall f \in Fl(u), v \in NE_S, q_f$ source of $f; \forall u \in NE_{VL}$
map_sink:	$new(u,v) \geq new(d_f, v)$	$\forall f \in Fl(u), v \in NE_S, d_f$ sink of $f; \forall u \in NE_{VL}$
req_fmin:	$\sum_{w \in NE_S} (flow_{r_V}(f,v,w) - flow_{r_V}(f,w,v)) \geq new(q_f,v) req(u,r_V,s) - new(d_f,v)\infty$ $\forall f \in Fl(u), v \in NE_S, r_V \in R_f; \forall u \in NE_{VL}, s = \texttt{minimum}$	
req_fmax:	$\sum_{w \in NE_S} (flow_{r_V}(f,v,w) - flow_{r_V}(f,w,v)) \leq new(q_f,v) req(u,r_V,s) + new(d_f,v)\infty$ $\forall f \in Fl(u), v \in NE_S, r_V \in R_f; \forall u \in NE_{VL}, s = \texttt{maximum}$	
req_fconst:	$\sum_{w \in NE_S} (flow_{r_V}(f,v,w) - flow_{r_V}(f,w,v)) = new(q_f,v) req(u,r_V,s) - new(d_f,v) req(u,r_V,s)$ $\forall f \in Fl(u), v \in NE_S, r_V \in R_f; \forall u \in NE_{VL}, s = \texttt{constant}$	

Link Allocation:

exp_out:	$\sum_{w \in NE_S} flow_{r_S}(f,v,w,r_V) \leq alloc_{r_S}(u,v,r_V)$	$\forall f \in Fl(u), v \in NE_S, r_V \in R_f,$ $r_S \in R_S, \forall u \in NE_{VL}$
exp_in:	$\sum_{w \in NE_S} flow_{r_S}(f,w,v,r_V) \leq alloc_{r_S}(u,v,r_V)$	$\forall f \in Fl(u), v \in NE_S, r_V \in R_f,$ $r_S \in R_S, \forall u \in NE_{VL}$
direction:	$flow_{r_S}(f,v,w,r_V) \leq new(u,v) cap_{r_S}(v,w)$	$\forall f \in Fl(u), (v,w) \in NE_S^2,$ $r_V \in R_f, r_S \in R_S, \forall u \in NE_{VL}$
relate_f:	$\sum_{w \in NE_S} flow_{r_S}(f,v,w,r_V) + flow_{r_S}(f,w,v,r_V) \geq new(u,v)$	$\forall f \in Fl(u), \forall u \in NE_{VL},$ $v \in NE_S, r_V \in R_f, r_S \in R_S$

Migration:

new:	$\sum_{v \in NE_S} old(u,v) \geq mig(u)$	$\forall u \in NE_V$
migrated:	$old(u,v) - new(u,v) \leq mig(u)$	$\forall u \in NE_V, v \in NE_S$

Figure 5.2: **Embedding constraints for linear Mixed Integer Program. Explanations are given in the text.**

max_load denotes the maximal load over all resources and is defined in the constraints of the MIP. This dual load approach is required to compensate for variation in availability of different resources: Minimizing only *max_load* would optimize only the scarcest resource and hence leave overly high slack in other resource allocations. Minimizing individual $load(r_S)$ avoids unnecesary resource allocations, but (again

numerically) overrules *max_load* as prime factor. Therefore, the constant factor c is required to balance between overall and individual load.

5.4.2 Constraints

The embedding must fulfill various type, capacity, and other consistency constraints, see Figure 5.2 for a complete constraint list.

Nodes: This constraint category is used to ensure that each VNet node is mapped to an appropriate substrate node. In contrast to links, we do not map nodes to multiple substrate elements, and hence Constraint `map_node` is necessary to guarantee a unique mapping location. At the location where the node is mapped (and only there!), resource requirements must be fulfilled (Constraint `set_new`). Depending on the substrate resource type (`minimum`, `maximum`, or `constant`), the resource constraints are imposed in a different manner (Constraints `req_min`, `req_max`, `req_con`).

Mapping: The mapping constraints ensure that the substrate element has sufficient capacity (Constraint `ne_capacity`) allocated. If resources are shared amongst substrate elements, we need to check against the capacity of the resource itself (Constraint `capacity`). In order to limit link split-ups, we set a minimal resource allocation unit (Constraint `relate_V`). Moreover, virtual elements hosted must be of the correct type (Constraint `allowed`). Constraint `load` and Constraint `max_load` define the load of a resource (i.e. the fraction of its capacity used) and the maximum of all individual resourc loads, respectively.

Resource-Variable Relation: This set of constraints deals with the relation between the resource types r_S that host resources of type r_V. In our mathematical program, we assume a linear relation, which is given by the constant factor $prop(r_V, r_S)$ (Constraints `resource` and `flow_res`).

Links: Mapping links is similar to mapping nodes, and hence, several constraints apply also to links. However, in contrast to nodes, links may be mapped to more than one substrate element (as one or several paths). Shared communication channels need to allocate resources to satisfy their requirements with respect to every virtual node pair connected. In order to calculate allocations in this case, links are expanded into a set of flows, as described earlier. Clearly, each virtual link must be mapped to at least one substrate element (Constraint `map_link`). Sources and sinks of the expanded flows definitely are part of this mapping: (Constraints `map_src` and `map_sink`). Note that this allows to find a valid mapping even for pure local links, i.e. if all VNodes are mapped to a single SNode. As a first step simplification, we assume that pure local links require only nominal resources, considering only resource allocation

corresponding to $min\_alloc_{r_V}$[3].

The multi-path propagation of each flow f must satisfy flow preservation, except for the source and sink element. The constraint depends on the value type (`minimum`, `maximum`, or `constant`): In case of a minimum type, the net flow of a given resource type must be a least the requested resources at the source and preserved otherwise. If the substrate element is the sink, the flow preservation invariant is suspended and the constraint becomes fulfilled trivially. To implement a corresponding selector, multiplication by a sufficiently large number (e.g., slightly larger than the maximal amount of involved resources, here simply represented by ∞) is used in the subtrahend. This yields the desired tautology (see Constraint `req_fmin`). The Constraints `req_fmax` and `req_fconst` are defined analogously.

Note that it is not possible within the MIP to strictly ensure maximum or constant bandwidth in combination with half-duplex links. This is due to conceptual reasons discussed in Subsection 5.5.4.

Link Allocation: The substrate resource r_S allocated for a virtual u on a substrate element v is the maximum of the r_S required for every single of u's flows. (Constraints `exp_out` and `exp_in`) ensure that these resources are allocated on sources and destinations of the respective flows. Constraint `direction` enforces direction specific capacity constraints on full-suplex substrate resources.

Migration: Our program allows us to migrate already embedded VNet elements to new locations, if the reconfiguration costs are amortized by the more efficient embedding. The migration constraints set the migration flag $mig(u)$ if[4] the mapped element is not new (Constraint `new`) and was previously embedded at a different location, where it was removed (Constraint `migrated`).

5.5 Extensions

This sections discusses possible future extensions to the above MIP. More specifically, in the following we sketch extensions to remedy our first-step simplifications with respect to link migration. We then discuss the incorporation of interface bottlenecks and the modeling of distributed virtual node embedding. We show why it is not possible to guarantee maximum and constant bandwidth constraints for generic half-duplex scenarios via the MIP. Finally, we sketch extensions allowing for latency constraints as well as more granular control regarding placement constraints.

[3] This can be extended trivially by adding a variant of constraints req_* for links u, where $new(u, v)$ is replaced by $new(u', v)$ for all VNodes u' connected to u

[4] and only if, whenever migration costs are relevant - i.e., > 0, and minimized in the objective function

5.5.1 Link migration costs

So far, we considered only simplified link migration costs. This subsection outlines modifications to relax this restriction.

The modeling of contract penalties can be added by extending the $mig(u)$ matrix to a $mig(u, v)$ matrix. This allows for a more granular formulation of modified virtual link segments rather than entire links. If a new constraint enforcing $new(u, v) > mig(u, v)$ is added, migration-maximizing objective functions can be used. This removes the remaining ambiguity of the `migrated` constraint where both $old(u, v)$ and $new(u, v)$ are 1.

In a case, in which all endpoints happen to change location at the same time, we assume a VROOM scenario in our MIP, in which temporary tunnels are spawned between migration source and destination positions. The cost for these tunnels are added to host migration costs. As an alternative, one may merely interconnect old and new virtual link u segments. In this case, the interconnection is modeled by an additional set of flows $f_{x_{[up,down]}}$ running between the old location of any one connected node to the new location of any other connected node. Adding the respective allocations temporarily ensures connectivity independent of actual migration switchovers of the endpoints. If a sum of $new(f_{x_{[up,down]}}, u)transit(u, v)$ is added to the objective function, its path will be optimized by the solver. Moreover, a constraint $new(f_{x_{[up,down]}}, u) \geq 1 - old(u, v) - new(u, v)$ results in charges being added only for the truly temporary allocations[5].

5.5.2 Interface bottlenecks and parallel edges

If the topology turns out to be a graph with parallel edges, the MIP assumes multiple interface connections between two network elements to be abstracted by one reflecting their effective combined bandwidth. Furthermore, the MIP considers link resource bottlenecks reflected in network elements (e.g., links) rather than interface connections.

If interfaces form bottlenecks of link resources (e.g., an interface allows for less bandwidth than an attached link), we need to add interface capacities:

- Constraints `ne_capacity`, `capacity` and `load` need to be replicated for each edge, i.e., edge allocations $alloc_{r_S}(u, v, w, r_V)$ need to be correlated with interface capacities $cap_{r_S}(v, w)$.
- Objective function and `max_load` need to consider the new $alloc_{r_S}(u, v, w, r_V)$ variable matrix.

[5] for cost minimizing objective functions

If the interfaces are to be distinguished inside the MIP, parameters change to $(v, w)_i$, rather than v, w in the above modifications.

5.5.3 Distributed node embedding

Our MIP considers virtual nodes atomic entities. It thus does not explicitely support a distributed node embedding (*node split-up*) where e.g., processing or RAM resources are spread across several substrate network elements. However, it does support the colocation of partial or complete virtual topologies on one substrate node. We therefore argue that the problem of distributed node embedding can be reduced to the problem of placing atomic (partial) nodes and calculating paths for interconnections. This may be done via one of two approaches, which relate to our description language presented in Chapter 4:

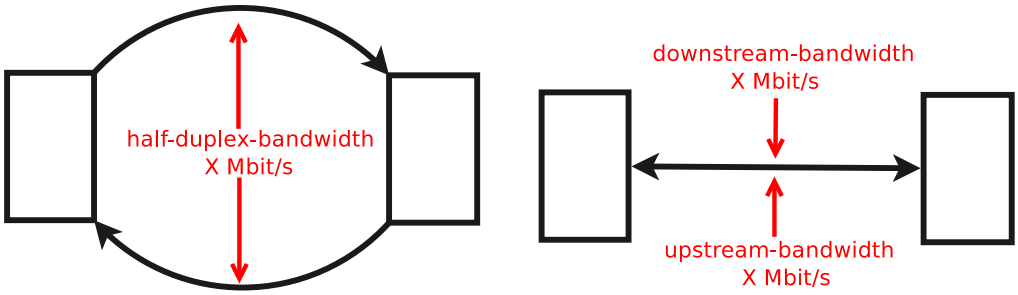
1. The substrate may be expanded into a topology of logical components representing possible placement solutions. Suppose for example that computing and storage can be mapped on two networked substrate components. Then a logical node can represent this specific combination and share corresponding resources with other logical instances. This approach abstracts the split-up via virtualization. As an example, consider an NFS server hosting virtual hard disks in an UL0 graph (see Chapter 4) removed in the UL1 graph, where the NFS share is modeled as resource shared between all substrate hosts that have mounted it.

2. The compound node can be split up into a topology of atomic subcomponent nodes and integrated into the network description. Since these can still be placed together on a single substrate node, distributed embedding is not enforced unless it is beneficial for the objective function.

The first version is only an option, if link resource overhead of distributed embedding is not an issue (e.g., because it is not considered in the VNet data plane). This is due to a complex overlap in shared substrate resources which is be reflected in the MIP.

5.5.4 Half-duplex limitations

Our MIP always guarantees a minimum amount of resources over and for half-duplex links. This is due to the construction of `exp_in` and `exp_out`. It supports $s =$ `maximum` and $s =$ `constant` requirements for many scenarios, if the objective function aims at a resource allocation minimization. But it cannot do so in generic scenarios. Support for half duplex links is always connected to a embedding mechanism that is supporting it. The design of such a mechanism is outside of the scope of this thesis.



(a) HD VLink to FD SLink (b) FD VLink to HD SLink

Figure 5.3: **Mapping (a) a half-duplex VLink onto a full-duplex, and (b) a full-duplex VLink to a half-duplex SLink.**

This becomes apparent when considering the following two scenarios depicted in Figure 5.3: In (a), provisioning of a minimum half-duplex bandwidth of X MBit/s on a full-duplex substrate link is coupled to a reservation of resources in both directions. Without aforementioned embedding support, the virtual half-duplex links can be used into both directions simultaneously. In (b), the situation is reversed. To provision X MBit/s per direction, 2X MBit/s have to be reserved in the substrate, which can be used simultaneously for unidirectional traffic.

5.5.5 Latency support

Constraints in the **Links** and **Link Allocation** categories can serve to calculate resources which are represented as commodity flows. However, the above MIP requires a notion of a path to compute resources, such as latencies, which are distributed along a path. There are two ways to introduce such a path notion:

1. On the description level: Every virtual link can be split up and mapped to a set of links with partial resources of desired granularity in another OL type VNet description. The MIP in turn needs to be modified to disallow flow split-ups.

2. On the MIP flow level: Each flow $f \in Fl(u)$ can be split up into a set of atomic (unsplittable) subflows $f_i \in FParts(f)$. This approach allows for better resource allocations at the cost of a more extensive set of modifications, as partial flows may remain empty (i.e., to be not mapped).

After introducing such a path notion, computing flow latency corresponds to summing of all latencies of substrate elements the flow is mapped on. Every atomic flow needs to satisfy the latency constraints.

Modifications following the second approach are sketched in Figure 5.4: The first part introduces the set of partial flows and correlates their substrate resource allocations with the corresponding virtual resources. The second part adds requirement verifications for virtual resource allocations of subflows to the regular checks on the flow level. Subflows are allowed to remain empty. Directional constraints and flow-link placement correlations are considered on the subflow level. Flows respect directions by transitivity and virtual links are placed wherever subflows exist. The fourth part renders subflows atomic by restricting the fanout on substrate vertices. The fifth formulates the aggregation of partial flows to full flows, and the sixth verifies latencies on the path (i.e., subflow) level. Capacity verification remains unmodified on the link level.

1. Introduce partial flows :
Expand all $f \in Fl(u), u \in NE_{\mathrm{V}}$ to atomic partial flows $f_i \in FParts(f)$.
Create `fpart_res` constraints on f_i (subflow) level analogue to `flow_res` constraints.

2. Formulate subflow problem rules, allowing for empty subflows :
Replicate all `req_fmax` and `req_fmin` constraints as `req_fpmax` and `req_fpmin` constraints on subflow level.
Substitute "$new(q_f, v)req(u, r_{\mathrm{V}}, s)$" by "0" in `req_fpmin` constraints.
Create `req_fpmax` constraints on subflow level for every `req_fconst` constraint present on flow level.

3. Correlate link placement and directional capacities with subflows :
Substitute "$new(u, v)$" by "$new(f_i, v, w)$", "f" by "f_i", and "$f \in Fl(u)$" by "$f_i \in FParts(f); \forall f \in Fl(u)$" in `direction` constraints.
Add `map_linkf` constraints:

`map_linkf`:	$new(f_i, v, w) \leq new(u, v)$	$\forall f_i \in FParts(f), f \in Fl(u), u \in NE_{\mathrm{VL}}, (v, w) \in NE_{\mathrm{S}}^2$

4. Render subflows atomic :

`new_fpart`:	$flow_{r_{\mathrm{S}}}(f_i, v, w, r_{\mathrm{V}}) \geq new(f_i, v, w)$	$\forall f_i \in FParts(f), f \in Fl(u),$ $(v, w) \in NE_{\mathrm{S}}^2, r_{\mathrm{V}} \in R_f, r_{\mathrm{S}} \in R_{\mathrm{S}}$
`no_splitpart`:	$\sum_{w \in NE_{\mathrm{S}}} new(f_i, v, w) \leq 1$	$\forall f_i \in FParts(f), f \in Fl(u),$ $(v, w) \in NE_{\mathrm{S}}^2$
`map_fpart`:	$\sum_{w \in NE_{\mathrm{S}}} new(f_i, v, w) = new(f_i, v)$	$\forall f_i \in FParts(f), f \in Fl(u),$ $v \in NE_{\mathrm{S}}$

5. Sum up partial flows to full flows :

`fpart_sum`:	$\sum_{f_i \in FParts(f)} flow_{r_{\mathrm{S}}}(f_i, v, w, r_{\mathrm{V}}) = flow_{r_{\mathrm{S}}}(f, v, w, r_{\mathrm{V}})$	$\forall f \in Fl(u), (v, w) \in NE_{\mathrm{S}}^2,$ $r_{\mathrm{V}} \in R_f, \forall u \in NE_{\mathrm{VL}}$

6. Check latencies :

`lat_fpart`:	$latecy(f_i) = \sum_{v \in NE_{\mathrm{S}}} new(f_i, u) latency(v)$	$\forall f_i \in FParts(f), f \in Fl(u), u \in NE_{\mathrm{VL}}$
`lat_min`:	$latency(f_i) \geq req(u, r_{\mathrm{latency}}, s)$	$\forall f_i \in FParts(f), f \in Fl(u), u \in NE_{\mathrm{VL}}$ $s =$ `minimum`
`lat_max`:	$latency(f_i) \leq req(u, r_{\mathrm{latency}}, s)$	$\forall f_i \in FParts(f), f \in Fl(u), u \in NE_{\mathrm{VL}}$ $s =$ `maximum`
`lat_const`:	$latency(f_i) = req(u, r_{\mathrm{latency}}, s)$	$\forall f_i \in FParts(f), f \in Fl(u), u \in NE_{\mathrm{VL}}$ $s =$ `constant`

Figure 5.4: **Additional constraints to model paths inside the MIP**

5.5.6 Constraint group handling

In our prototype, we handled incomplete descriptions outside of the MIP. In particular, *ConstraintGroups*, introduced in Chapter 4, are resolved prior to MIP formu-

lation[6]. ConstraintGroups allow for multiple *Feature* values, that enable or disable placements of VNEs on groups of substrate elements sharing this property. This is reflected in the MIP's $suit(u, v)$ matrix. Every *Feature* is assigned a tentative value. Failure to find a solution on the MIP level is handled by backtracking. By adding flexibility to the MIP for handling suitability issues, it may be possible to remove backtracking steps.

A matrix $configuration(u, c)$ can model configuration group memberships for virtual components u and configuration c. A new constraint would need to compare a variable matrix $suit(u, v)$ to a sum of acceptable property group memberships. It allows placement only if at least one compatible group c is selected for u. A new constraint is needed to compare $configuration(u, c)$ against compatible property p memberships $prop(u, p)$ and incompatible property memberships $1 - prop(u, p)$ for each given group g.

5.6 Evaluation

This section reports on our experience with the prototype implementation described in Chapter 8. We conducted experiments for three basic scenarios:

VPN: In the *VPN scenario*, the locations of the virtual nodes of a VNet are fully specified. In contrast to typical VPN networks however, resources are reserved along the paths connecting the VPN terminals (admission control and traffic shaping to ensure QoS). It essentially boils down to a network flow problem. Moreover, computational and storage resources can be specified at the terminals.

DC: The *data center scenario* describes the other extreme where the virtual nodes have full placement flexibility and can be mapped to arbitrary locations.

OC: The so-called *out-sourcing/cloud scenario* is situated between the two extremes modelled by the static VPN and the fully flexible DC scenario. A VNet consists of some virtual nodes with fixed locations (e.g., branches of the company and access network) while other virtual nodes (providing, for example, storage or computation) are "out-sourced", e.g., to a cloud or a data center, and have a flexible location.

In the following, we first state and motivate our evaluation goal. We then describe the data basis along with the environment of our evaluations, and proceed by presenting the results for each of the above scenarios.

[6] with the exception of co-location constraints, which are added as constraints to the MIP

5.6.1 Evaluation goal

Mathematical Solvers, such as CPLEX, are designed to deliver the proven optimal result to a problem with respect to the stated objective function. The consequence of this insight is that usual qualitative evaluation goals are only of limited use in the context of this chapter.

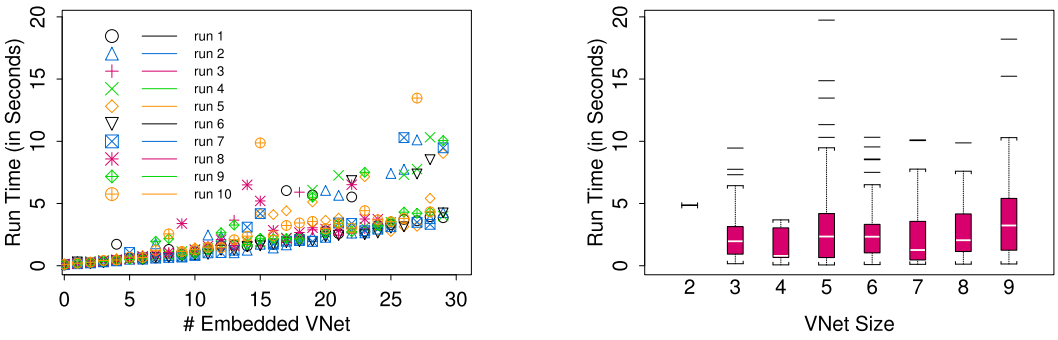
This becomes apparent when considering the fact that the possibility of migration (i.e., reallocations) prevents the rejection of a VNet based on resource placement inefficiency. Moreover, the output result is always - by design of the solver - optimal with respect to stated intentions (that is: the objective function). Traditional evaluations considering, e.g., 'cumulative cost over time' or 'rejection rate' thereby would not measure the quality of the formalization, which we want to evaluate. Instead, they come down to an evaluation of appropriateness of stated objective functions or request rejection policies in the context of considered scenarios.

Both may be of interest in subsequent research efforts, but are considered out of scope with respect to this thesis. We therefore turn to the most pressing topic of runtime. Our experiments are designed to provide insight into two major questions: To which extent can solving a NP-hard problem optimally be expected to scale in real scenarios? And what is the influence of parameters such as the percentage of freely placeable resources?

5.6.2 Data basis and environment

In order to model the physical substrate network, we extracted *Rocketfuel topologies* [117]. Connected subsets of these graphs are also used to describe the topology of the VNet requests. For the OC use case, the virtual nodes of the VNets are partitioned into freely allocatable cloud resources (CR) and fixed access points (AP) (e.g., connection points to corporate subnets of the requesting entity). In the DC use case all virtual nodes exhibit full placement flexibility. The VPN use cases is included implicitely in the OC evaluation.

Unless stated otherwise, the following configuration holds for the experiments: Substrate network elements feature capacity for fifteen virtual network elements. No placement preferences are given, i.e., migration penalties are destination independent and weights are 1 for all virtual network elements The embeddings are optimized with the maximal load minimization objective function. As a solver, IBM's standard `CPLEX` software is used in deterministic mode with a limit of six concurrent threads on a 8-core Xeon server running at 2.5GHz. MIP instances are created by our prototype implementation. Obviously irrelevant variables or constraints (e.g., allocation variables for places where no suitability is given) are not formulated.



(a) Mapping of nth VNet (b) Mapping of n-node VNet

Figure 5.5: **Run time without migration for multiple VNet placement.**

5.6.3 Out-sourcing Scenario

In the out-sourcing scenario, the virtual nodes fall into two categories: a set of fixed APs and a set of freely placeable CRs. Concretely, for each VNet we chose (uniformly at random) between one and three flexible VNet nodes and between one and seven fixed access nodes. The percentage of flexible nodes in the VNet is captured by the variable $freedom \in [0,1]$. Note that $freedom = 0$ is our VPN scenario. For our experiments, we use a substrate network of twenty-five nodes, and we iteratively place incoming VNet requests. Evaluations are repeated ten times. All VNet requests are accepted as long as resources are available. We study scenarios with and without migration.

Figure 5.5(a) shows the runtime (real time, in seconds) required to map VNets of different size iteratively (one after the other, sorted on the x-axis) in a scenario without migrations. Figure 5.5(b) shows results of the same experiment as a function of VNet size. There are several takeaways from these experiments: First, we observe that the embedding times are small (never exceeding 14 seconds). Moreover, depending on the load on the substrate network (the number of already embedded VNets and the VNet size), the runtime increases slightly. The data also exhibits a relatively high variance, which can be explained by the randomized nature of the to be embedded VNets (in terms of size and nature).

The run times generally increase if we enable the option to migrate, see the logscale plot in Figure 5.6, although there are instances where it is roughly the same. This is to be expected as migration increases flexibility and therefore the complexity of the MIP. An interesting feature of integrating migration support is that we can at any time check if a subset of the resources, e.g., half of the network is sufficient to

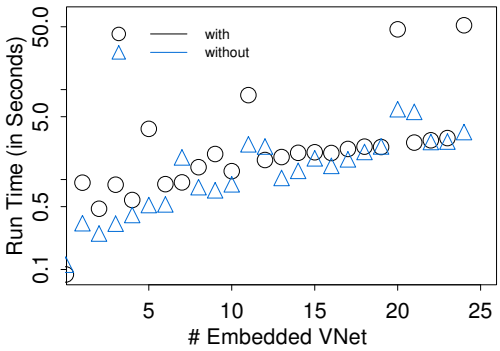


Figure 5.6: **Run time with and without migration (sample run).**

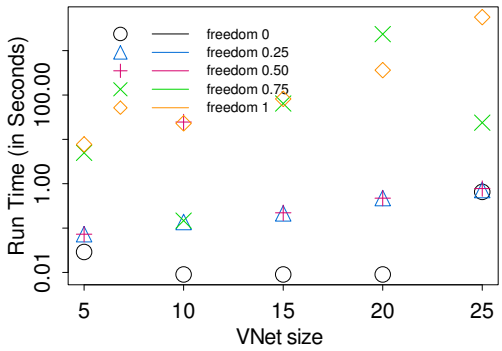


Figure 5.7: **Run time for different degrees of freedom (sample runs).**

fulfil the demand. In the above cases such a run takes on average 2.73 seconds with a standard deviation of 0.42 seconds.

As the above experiment suggests, a major parameter that determines the time complexity of the embeddings is the freedom of the node placement. We conducted a series of experiments where the VNet size and the proportion of CRs, (i.e., the variable $freedom$) varies. The findings are summarized in Figure 5.7. While the results show a certain amount of jitter (consider Section 2.2.4), they still confirm this dependency. Interestingly, despite the flexibility of the VNets and the existing load on the substrate, the run times are still in the range of several minutes.

We can conclude that although the option to migrate and the placement flexibility effect the execution times, optimal solutions for relatively large problems are feasible and can be computed in reasonable time. Moreover, as the run times without migration support are lower than their counterparts, hybrid designs, where incoming VNets are first placed ad-hoc, and persisting ones are optimized regularly (by an offline, background process) are attractive.

5.6.4 Data Centers

The data center use case exhibits the highest flexibility and hence constitutes, in some sense, the "worst case" scenario (in terms of embedding complexity). In order to quantify the impact of the substrate network size, we calculated mappings for a single twenty-five CR VNet on substrates of different sizes. In one set of experiments, we calculated the optimum, in another we emphasized feasibility. For the

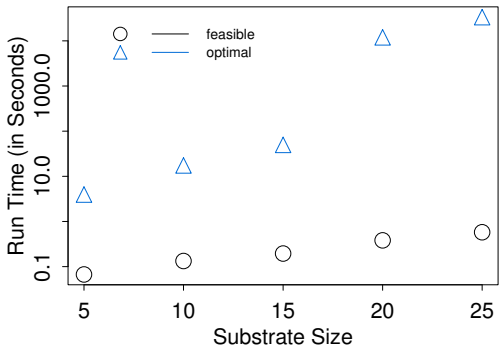


Figure 5.8: **Run time vs. substrate size in DC scenario: optimal vs. feasible solution (sample runs).**

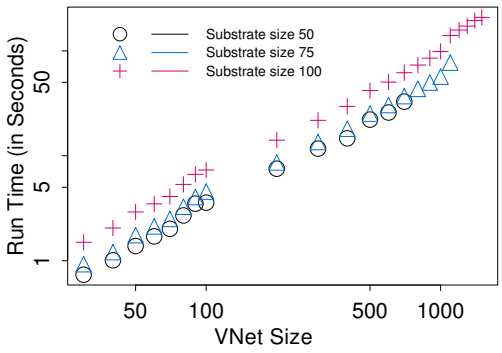


Figure 5.9: **Run time vs. VNet size for different substrate sizes (sample runs).**

latter experiments, we turned off the multi-threading and parallelization features of `CPLEX`.

Figure 5.8 studies the price of optimality: the comparison of the run times for optimal and feasible (i.e., first possible) VNet embeddings shows that while the performance of both depends on the substrate size, optimal solutions may result in an order of magnitude higher run times. For a substrate network of around twenty nodes, we can still expect an optimal solution within hours. Recall, this is a worst case scenario with respect to (full) flexibility. Nonetheless, it shows the importance of investigations into approximation approaches.

Finally, we examine feasibility in detail and report on an experiment studying the run times as a function of the VNet size (cf Figure 5.9): the loglog plot indicates that the runtime grows linearly for larger network sizes. Even VNets with thousands of elements can be embedded within minutes in a substrate of almost one hundred nodes.

As a final remark, note that although our run times for the data center embeddings are higher than in the out-sourcing use case, the conclusion that VNet optimizations are to time-consuming in data centers may be wrong in particular, as approximations can help to reduce the runtime drastically. Our experiments are overly conservative in the sense that in reality, we expect that VNet requests for real data centers to be homogeneous, and are issued to, e.g., computing grid environments which are typically likely to be homogeneous as well. This homogeneity facilitates a range of

optimizations, e.g., by aggregating entities into larger meta-nodes; this reduces the number of elements (i.e., variables) in the problem.

5.7 Related Work

There has been a significant interest in virtual networks over the last years. The reader is referred to the recent surveys [46] and [71]. The work described here is conducted in the context of our network virtualization project where we develop a prototype implementation.

Network embedding problems have already been studied in various settings (e.g., in the context of circuit-design, which is however quite different and not discussed further here). Note that the virtual network embedding problem is different from classic *VPN embedding* or *multi-flow* problems in the sense that the node placement is not given but subject to optimization as well. This additional degree of freedom renders the problem more complex. Indeed, many variants already of much simpler virtual network embedding problems are computationally hard: Even if all virtual network requests are given in advance, the offline optimization problem with constraints on virtual nodes and virtual links can be reduced to the NP-hard *multi-way separator problem* (e.g., see [20] for a survey). Thus, there is a large body of literature on *heuristic solutions*: For example, Fan and Ammar [62] study dynamic re-configurable topologies to accommodate communication requirements that vary over time, Zhu and Ammar [140] consider virtual network assignment problems with and without reconfiguration but only for bandwidth constraints, Ricci et al. [104] pursue a simulated annealing approach, and Lu and Turner [86] seek to find the best topology in a family of backbone-star topologies. Many approaches in the literature fail to exploit the flexibility to embed virtual nodes and links simultaneously and solve the two mappings sequentially (e.g., RW-MaxMatch [85]), or in multiple steps with backtracking (e.g., [85]), which entails a loss of efficiency [52]. To deal with the computational hardness, Yu et al. [137] advocate to rethink the design of the substrate network to simplify the embedding, e.g., by allowing to split a virtual link over multiple paths and perform periodic path migrations. The focus of the work by Butt et al. [42] is on re-optimization mechanisms that ameliorate the performance of the previous virtual network embedding algorithms in terms of acceptance ratio and load balancing; their algorithm is able to prioritize resources and is evaluated by simulations. Finally, virtual network embeddings have also been studied from a distributed computing point-of-view [76].

Our work seeks to combine virtual networks with storage and computation to enable virtual networks. An emphasis is put on *generality* and *quality* of the embeddings. We believe that the mathematical programming approach we pursued has many advantages, as it allows for a simple replacement of the objective function, and as state-of-the-art and optimized solvers can be used to find not only optimal but

also approximate or heuristic solutions. (There is no need to reinvent, e.g., new pruning heuristics for each embedding problem variant; often such heuristics are also unlikely to be faster than the sophisticated algorithms incorporated into `CPLEX` or `lpsolve`.) We are only aware of two embedding approaches related to virtual networks for which a mathematical program solves both node and link placement: Kumar et al. [83] describe an approach to solve a Virtual Private Network tree computation problem for bandwidth provisioning; flexible virtual node placements are not possible. Chowdhury et al. [44] present an linear embedding program and pursue a relaxation strategy, applying randomized and deterministic rounding to find approximate solutions. The presented graph extension approach supports exact solutions for placements where interconnected virtual nodes do not share candidate substrate nodes, as it would add bogus resources otherwise. Hajjat et al. [72] calculate reconfigurations in the context of enterprise applications under the constraint of communication costs after migration. Costs of the migration itself are not considered. Guo et al. [70] describe a clustering approximation placing nodes and links successively in Data Center environments. While migrations are supported and minimized, their costs per se are not formulated. For an online and competitive algorithm see [59]. Houidi et al. [76] describe a distributed algorithm for mapping VNets, which is based on a decomposition into hub-and-spoke clusters. Also, several challenges of embeddings in wireless networks have been identified by Park and Kim [95].

To the best of our knowledge, there is no algorithm to embed VNet like networks in a manner whose generality and flexibility is close to ours. In particular, none of the solutions above can handle all the heterogeneous links occurring in practice and map, e.g., a broadcast link onto a set of asymmetric and full-duplex links. Also, we are not aware of any algorithm which, e.g., allows to capture CPU load induced due to packet rates of the flows in a VNet; finally, we believe that the support of cost-aware migration is crucial, as the dynamical aspects lie at the heart of network virtualization.

5.8 Summary

In this section, we formalized the virtual network mapping problem as a dynamic multicommodity flow problem and designed a mixed integer program solving the mapping. Our solution facilitates easy changes to the optimization criteria, and thus can be easily adapted to the need of different entities with different goals. We support handling of real-world network topologies, including shared communication channels with multiple endpoints and shared resources. Our model of migration costs allows us to weight embedding optimizations against costs implied by the respective transitions.

Furthermore, we sketched possible extensions and improvements of our MIP, allowing for distributed node embedding or the incorporation of latencies. We evaluated the solution's runtime with respect to three characteristic use scenarios. We find that while runtimes may be sufficient for optimization of long term VNets, investigations into approximate solutions are sensible especially for scenarios with limited allowable solving time or high degree of freedom with respect to placement. These investigations are addressed in Chapter 6.

Chapter 6

Approximation approaches

As shown in Chapter 5, the runtime of our MIP approach is acceptable for the outsourcing scenario, but does not scale that well for scenarios with high degrees of placement freedom. To tackle the runtime problem, one can use solvers to approximate on MIPs. This can be beneficial since the solvers deliver a solution which is within a guaranteed bound of the optimal one.The question is to what extent which approximation approach helps in improving the runtime and how it affects the quality of found solutions[1].

We start by defining a quality function we subsequently use for qualitative comparison of solutions. We identify relevant parameters for the runtime. Next, we then evaluate the effect of solver parameters. Subsequently, we examine the effect of problem formulation modifications and clustering. We conclude with a short discussion of the results.

6.1 Quality and parameters

We first define a quality function that will subsequently be used to assess approximation costs. We then start our investigations by verifying to what extent problem parameters (especially network topology) correlate with runtime changes.

6.1.1 Quality function

In this Chapter, we take the position of a provider trying to host the maximum number of virtual resources on his substrate. We therefore run our experiments with the resource minimization objective function (see Section 5.4).

As in Chapter 5, rejection policies and similar higher-level aspects are considered out of scope. We therefore again do not directly consider aggregate quality metrics, such as rejection rate. However, it is important to notice, that in the scope of our fixed scenario, in which the goal is efficiency in resource usage, and in which every embeddable request will be accepted, our basic quality metrics correlate directly

[1]Part of the work in this chapter was done in the context of [68]

to the rejection rate. Overallocation of substrate resources reduces the number of supported virtual elements, and thereby increases the rejection rate.

Consequently, we subsequently use one of two quality measures to capture the effect of approximations in terms of runtime and embedding quality: Either (*#VNodes*) by the number of accomodated VNodes in experiments where the substrate is filled with subsequent requests, or (Q_π) by the amounts of resources required for VNet hosting to compare individual mapping qualities. The latter (Q_π) is defined as follows:

Formally speaking, a network graph $G = (V_G, E_G, r)$ consists of a set of $|V_G|$ nodes $V_G = \{v_{G_1}, v_{G_2}, \ldots, v_{G_{|V_G|}}\}$, a set of $|E_G|$ links consisting of tuples $e_{G_h} = (v_i, v_j) \quad i, j \in [0, |V_G|]; \quad h \in [0, |E_G|]$ and a function $r : V_G \cup E_G \to \mathbb{R}$.

A substrate S is a network graph where the r function describes the total resources on each node and link. A VNet VNR is also a network graph, but here the r function represents the resource requirements for the virtual network nodes and links. A placement $\pi_S^{VNR} = (LM_\pi, NM_\pi)$ consists of a node mapping $NM_\pi : V_{VNR} \to 2^{V_S \times \mathbb{R}}$ and a link mapping $LM_\pi : E_{VNR} \to 2^{E_S \times \mathbb{R}}$. For this functions there exist inverted functions:

$$NM_\pi^{-1} : V_{VNR} \times V_S \to \mathbb{R} \tag{6.1}$$

$$NM_\pi^{-1}(v_{VNR_n}, v_{S_m}) = \begin{cases} 0 & \forall r \in \mathbb{R}.(v_{S_m}, r) \notin NM_\pi(v_{VNR_n}) \\ r & \exists r \in \mathbb{R}.(v_{S_m}, r) \in NM_\pi(v_{VNR_n}) \end{cases} \; n \in [1, |V_{vnr}|], \; m \in [1, |V_S|]$$

$$LM_\pi^{-1} : E_{VNR} \times E_S \to \mathbb{R} \tag{6.2}$$

$$LM_\pi^{-1}(e_{VNR_n}, e_{S_m}) = \begin{cases} 0 & \forall r \in \mathbb{R}.(e_{S_m}, r) \notin LM_\pi(e_{VNR_n}) \\ r & \exists r \in \mathbb{R}.(e_{S_m}, r) \in LM_\pi(e_{VNR_n}) \end{cases} \; n \in [1, |E_{vnr}|], \; m \in [1, |E_S|]$$

The quality Q of a specific placement π for a given VNet is

$$Q_\pi = \frac{\sum\limits_{v \in V_{VNR}} \sum\limits_{v' \in V_S} NM_\pi^{-1}(v, v')}{\sum\limits_{v \in V_{VNR}} r(v)} + \frac{\sum\limits_{e \in e_{VNR}} \sum\limits_{e' \in E_S} LM_\pi^{-1}(e, e')}{\sum\limits_{e \in E_{VNR}} r(e)} \tag{6.3}$$

As host resources are assumed to require the same number of resources in embedding regardless of their placement, the quality function reduces to:

$$Q_\pi = 1 + \frac{\sum\limits_{e \in e_{VNR}} \sum\limits_{e' \in E_S} LM_\pi^{-1}(e, e')}{\sum\limits_{e \in E_{VNR}} r(e)} \tag{6.4}$$

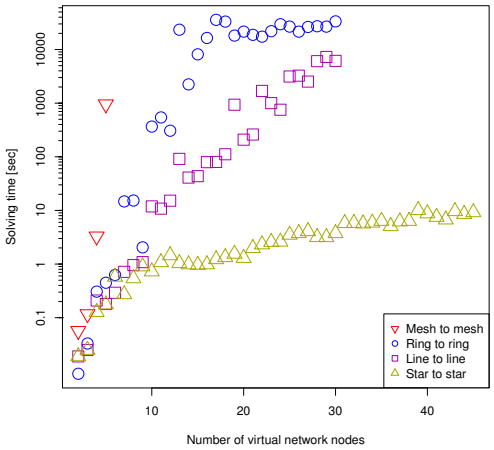


Figure 6.1: **Solving time for the same topologies on substrate and virtual network**

6.1.2 Parameters

We again investigate the impact of topologies on runtime. To this end, we map some archetypical topologies (line, star, ring, full mesh) of 2 to 45 nodes onto substrate topologies of the same type with a runtime limit of 10h. The substrate consists of 15 nodes. Each features capacity for 3 virtual nodes. The capacity of VLinks is 1/30 of the capacity of SLinks. In the following, we denote such a capacity configuration as *3n_30l*. We assume a position of a provider who wants to accomodate as many VNets as possible and (unless stated otherwise) therefore perform experiments with the resource minimizing objective function.

In the first experiment, we map VNetsof the same type on substrate topologies (i.e., a star on a star, a line on a line, etc.). Figure 6.1 clearly shows an impact of topology on runtime. While line, star, and ring topologies share a similiar ratio of nodes and links, star topologies are the only ones to be easily solved. This can be explained by the characteristics of the star experiment. Substrate nodes in the line and ring experiments are mostly equivalent to each other, while the center node of a star represents a bottleneck position. Moreover, placing a virtual star center on an exterior substrate node will likely incur additional link costs. We reason that the solver is able to detect these characteristics and use it to prove optimality of solutions more quickly. We used a 10h time limit. The mesh experiment exceeded the time limit at VNet size 6.

Next, we map different VNetsto different substrate topologies. Considering these cross mappings, Figure 6.2 confirms our analysis, since experiments involving star

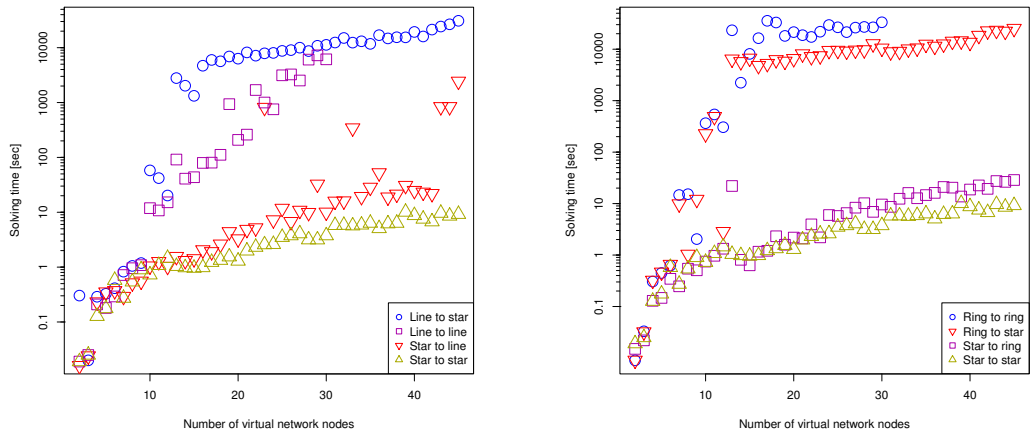


Figure 6.2: **Solving time for different topologies on substrate and virtual network**

topologies have runtime advantages. The only exception to some extent is the line-to-line experiment. It runs quicker for some instances, but exceeds the time limit at size 31. Line-to-star is solvable for larger instances. Since this is the only exception, we presume that symmetry between virtual and substrate topologies has an effect. However, it is weak in comparison to other effects. Given the limited evaluation range for mesh topologies, we omitted this topology type from subsequent experiments.

Figure 6.3 considers the presolving step. It shows that there is no significant reduction of variables for either problem instance which would explain the difference in runtime. Therefore, the difference must lie in the solving path chosen by the solver. Closer inspection of how the solver solves the problem instances (see Figures 6.4 and 6.5) yields three characteristic solving curves for boundary development.

In all inspected cases the solver finds an optimal integer solution within a short time frame. However, proving that the solution is optimal can take a long time reflected by the development of the lower bound in the relaxed problem instances. In the 24 VNode example (which finishes after mere 7s of solving time), the solver seems to find a good cutting plane and thus excludes a major portion of expansible nodes. It therefore is able to quickly show optimality w.r.t. the configured CPLEX default GAP parameter value of 10^{-4}. In the 23 VNode example, such a cutting plane is not found, and a large amount of nodes needs to be expanded in order arrive at the desired proof. In other examples like the 43 VNode diagram however, a good cut is found which brings the lower bound close, but not close enough w.r.t. the defined GAP parameter value to the optimal integer solution.

Unfortunately, as pointed out in [80], even minor details (such as the order in which

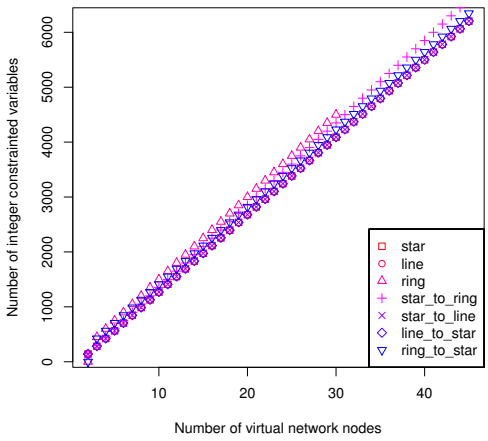


Figure 6.3: **Problem size after presolving**

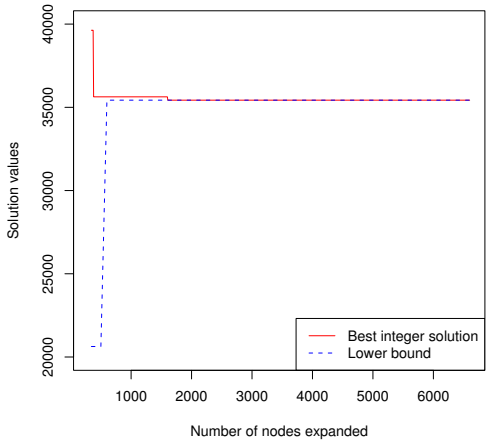
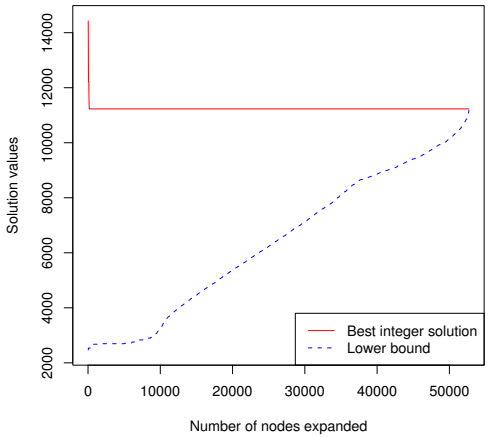


Figure 6.4: **CPLEX solving path for 43 VNodes (star to line)**



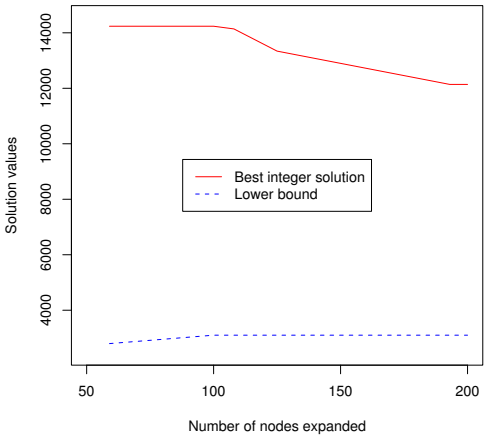


Figure 6.5: **CPLEX solving pathes for 23 (left) and 24 (right) nodes (star to line)**

constraints are formulated) impact on the solving path and generate major differences in runtime for specific problem instances. We therefore have to take this into consideration when evaluating jitter artifacts. In our experience, we find that good to optimal solutions seem often found at early stages and that proving optimality is a major runtime factor. We therefore proceed in the next section with an investigation of the effect of solver parameter modification (e.g., the configured GAP parameter value).

6.2 Solver related approximations

As solvers have a set of configurable parameters which can be used to generate approximate solutions. We address the effect of the most promising parameters in this section. More specifically, we consider solving limitations, such as the number of solutions or solving time and we investigate the effects of allowing for quality slack by GAP parameter modifications.

6.2.1 Limiting the number of solutions

While it may be considered a natural approximation to limit the number of solutions, we see more promise in other parameters. The reason for this is that in general we can neither predict how many solutions are generated within a time frame, nor how many solutions are found in total even for large problem instances. Moreover, we cannot predict in which way the parameter modification changes the solving path. It is likely to depend also on a series of additional factors, such as the number of parallel threads.

Limiting the number of solutions to a value > 1 therefore may not yield an effective runtime limitation at all (e.g., if the total number of explored integer solutions stays below the chosen value). In the previous initial experiments, we see a high number of test runs with only a limited number of explored integer solutions per run, rendering 1 the most sensible value. This illustrates that the solver in most cases specifically has to go out of its way to find an integer solution and implies two likely outcomes to experiments: Either the solver uses a depth first search to find a quick solution (yielding arbitrary quality solutions), or it uses a breadth first search to find the best solution (eliminating the runtime gain). It can be expected to chose its approach in a non-trivial fashion, preventing reliable predictions on the runtime. Given that we see high potential of runtime savings in the approach of removing the objective function (discussed in 6.3.1) without this uncertainty, we focus on the other approaches.

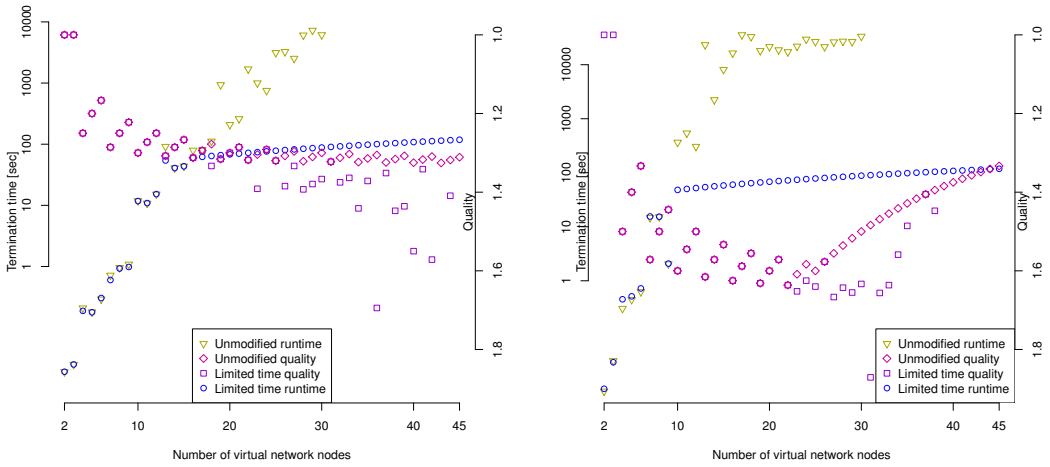


Figure 6.6: **Time limiting attempt for lines (left) and rings (right)**

6.2.2 Limiting solving time

Estimating time required to generate a "good" (or any) solution is hard, given that it is hard to predict the heuristic logic in chosing solving paths. Nonetheless, limiting solving time appears to be a realistic approach, as it has a guaranteed effect whenever runtime is an issue. Moreover, external factors like, e.g., embedding time constraints, can be trivially translated into parameter values.

We therefore evaluate its effect on the two test sets which show both potential for runtime improvements and previously yielded a sufficient number of results: "line on line" and "ring on ring". We did not modify any of other parameters. In addition, due to the regular structure of the scenario, we are able to calculate the optimal values for missing problem sizes. We chose a time limit which corresponds to the problem size. More specifically, we chose a time limit of $|V_s| + |V_{vnr}| + |E_s| + |E_{vnr}|$ seconds.

Both runtime and quality results are displayed in Figure 6.6. As expected, both experiments yield unmodified results as long as runtimes are below the time limit, and are bound by the timelimit thereafter. Although the differences in runtime quickly become significant, found solutions are still of good quality. A quality of 2 corresponds to a solution where on average every virtual link is mapped across a substrate link. The results are typically within a factor 2 of allocations necessary. While line topologies can be mapped for all sizes, the solver fails to solve some ring problems of size 36 and larger. This can be prevented by a better choice of time limits. Overall however, runtime improvements of a factor > 100 at limited quality degradation seem to make this approach an attractive choice.

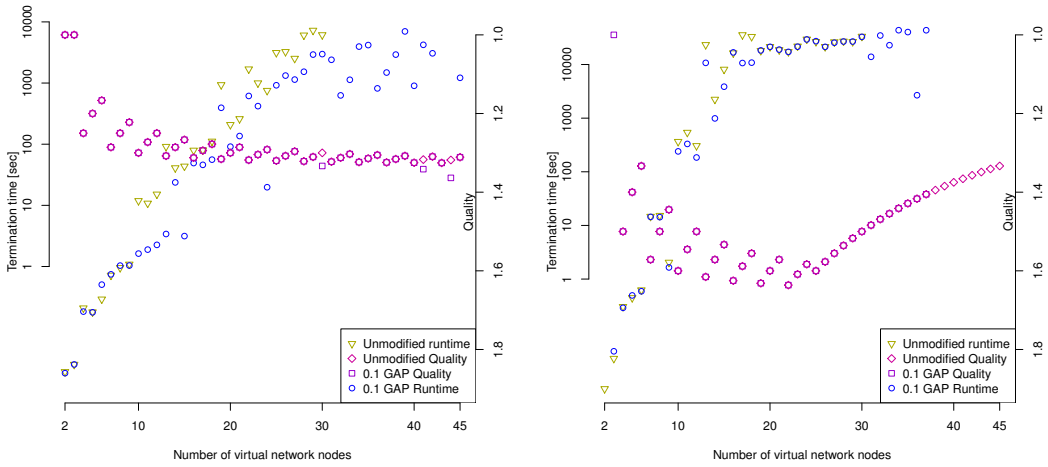


Figure 6.7: **Solving times for the mapping of a line to a line (left) and ring to ring (right) with gap** $= 0.1$

6.2.3 Increasing GAP parameter

An alternative approach also motivated by results shown in Figure 6.4 is to increase the GAP parameter value. This value configures the slack the solver is allowed with respect to optimality (i.e., the tolerance on what is considered optimal - at a value of 1, every solution is considered optimal). Increasing the tolerance can spare the solver a possibly long analysis of optimality when a solution of tolerable quality is found.

Figure 6.7 compares the runtimes and solution qualities of test runs with a GAP parameter value of 0.1 to these with an unmodified GAP parameter. Although the solver is allowed 10% slack in optimality, solutions are almost always optimal. While the effect is not consistent over all problem instances, runtime drops on average and previously not solvable instances become solvable. This confirms that the solver spends a considerable fraction of its runtime on proving optimality. Therefore, relaxing the GAP parameter is a viable element for reducing runtime.

6.2.4 Monitoring GAP development

Only in runs where the solver follows the patterns in Figure 6.4 or Figure 6.5 (left) will the GAP parameter setting have a considerable effect. In other cases, monitoring how the GAP develops will be beneficial. CPLEX reports on its current state in terms of best integer (BI) solutions, lower bounds (LB) found, as well as the resulting current GAP. Building a wrapper around CPLEX calls, we can thus observe how

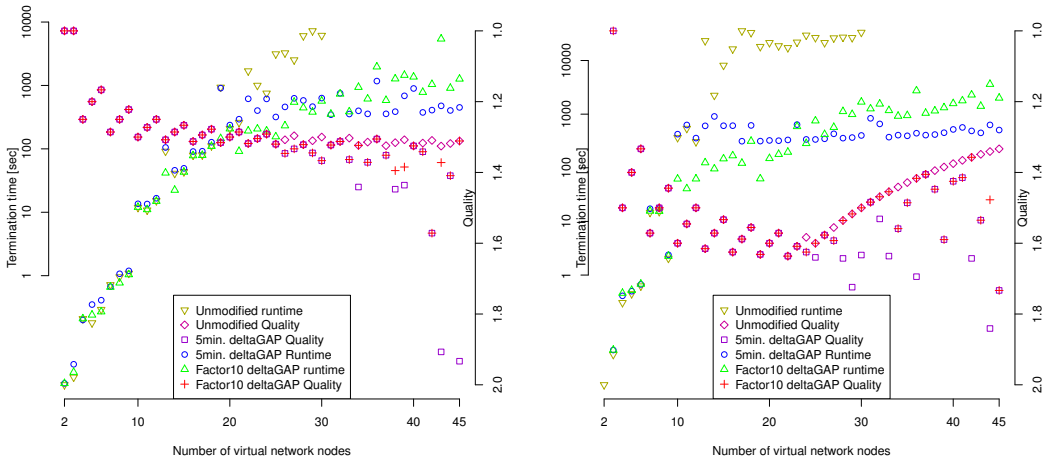


Figure 6.8: **Solving times for the mapping of a line to a line (left) and ring to ring (right)**

the GAP develops over time and stop the solver if either a timeout is reached or progress is too slow.

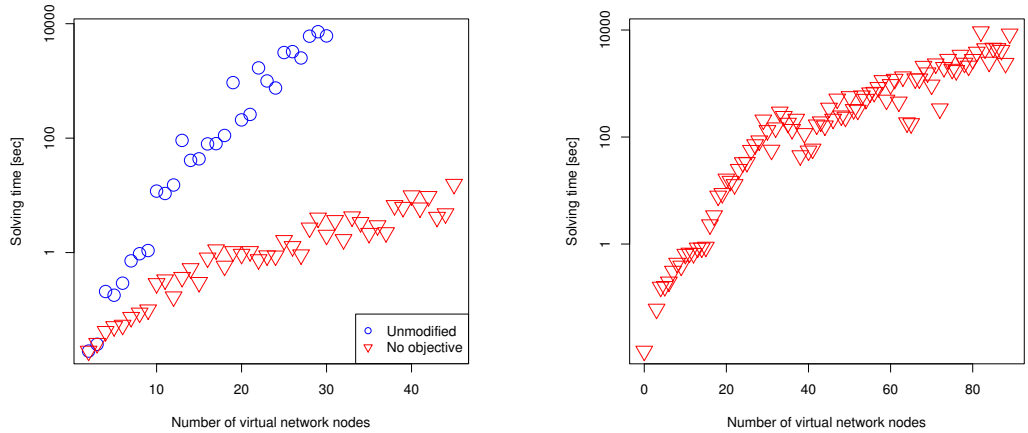
The wrapper allows the solver a certain amount of time Δt to improve the GAP by at least 5% after a integer solution has been found. If the improvement limit is reached, the timer is reset. We ran two experiments: one with a fixed time frame of 5 minutes and the other with 10 times the time required to find the first integer solution.

Figure 6.8 shows the results. The first observation is that the solver finds a solution to every problem instance at significantly reduced runtimes. Moreover, the instances that were previously solvable do not suffer from major quality loss. Most of the results are within a factor of 2 with respect to overallocated link resources. When the factor increases to approx. 3, the substrate is nearly fully utilised. This reduces when Δt was relating to solution finding time.

In summary, we see a slightly higher tendency towards overallocation when we monitor GAP, rather than setting the GAP parameter, but see a much more significant runtime advantage. Both approaches are promising options.

6.3 Problem formulation approximations

Aside from solver parameters, the problem formulation itself can be modified to find approximate solutions. More specifically, only a subset of the mapping aspects may be optimized and weights can be used to reduce the problem space symmetry.



(a) Mapping a line on a line (15 SNodes)

(b) Mapping a ring on a ring (50 SNodes)

Figure 6.9: **Solving times for 2-step mapping (timeout: 24h)**

6.3.1 Feasible solutions with optimized link placements

Following the idea of generating feasible placements rather than optimal placements one can split the process in two steps. The first step calculates node placements under given link constraints. The second step considers the nodes as fixed ($freedom = 0$), and merely solves the multicommodity flow problem to optimize resource allocation.

The goal of the first step is to find a feasible solution. Unfortunately, most solvers don't try to find any feasible solution quickly, but rather use a complex approach to find a "good" first solution. More specifically, open solvers, such as SCIP, as well as most probably closed solvers, such as CPLEX, follow roughly the following schema: Based on presolving logic, the solver first starts with a breadth-first search for a good solution. It avoids placing all elements as long as possible, gaining insight by exploring nodes with partial placements and cutting inefficient subtrees. This is the reason why the first found integer solution usually is already 'good', but it takes time to find it. By removing the objective function in the first step, we incite the solver to skip this phase and proceed with a depth-first search to find a solution quickly.

The second step can then be solved using a pure linear program (depending on allocation variable and specification requirements). In our experience even integer constrained problems with a freedom degree of 0 (by introducing constraints fixing the boolean variables) can be solved in a matter of seconds. Figure 6.9 shows the

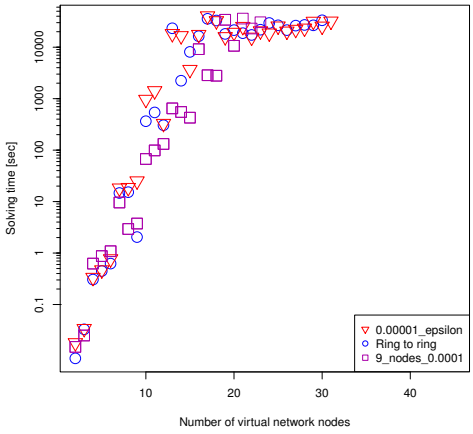


Figure 6.10: **Solving times for symmetry breaking experiments (runtime limit: 12h)**

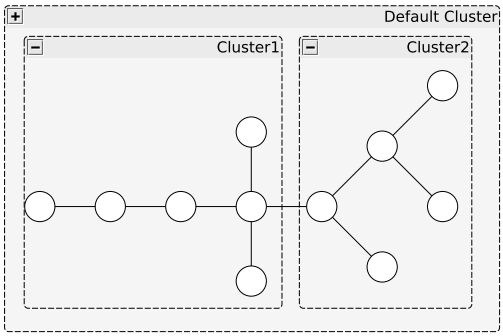


Figure 6.11: **Substrate for the experiment**

runtimes of the first step for two placement experiments. The left side compares runtimes with those of previous runs.

Motivated by the results, we attempted a placement of rings onto a ring substrate consisting of 50 SNodes (see Figure 6.9(b)). On the positive side, substantial runtime improvements are visible in both plots. Nonetheless, the ring experiments exceeded the time limit (24h in this specific case) before reaching placement of 100SNode rings, showing clear limits this approach. On the negative side, we need to face an arbitrary quality loss with respect to the solutions. The solver has a tendency to fill up SNodes before proceeding to the next (random) one, limiting the quality loss to a certain extent. This however cannot be generalized to arbitrary problems and objective functions. We therefore decided not to use this approach any further, as it clearly is of limited applicability.

6.3.2 Symmetry breaking

The next approach was motivated by the intuition that proving optimality in the presence of many solutions of equal costs can involve exploring an unnecessary high number of solutions. By adding a uniform random $\epsilon(u, v)$ matrix to the $weight(u, v)$ constant matrix, one can break symmetries and reduce the number of optimal solutions.

We ran a series of experiments mapping rings on rings. We thereby choose an $\epsilon(u, v)$ matrix not changing the set of optimal results (i.e., where the sum of weighted al-

locations for a resource does not warrant a suboptimal allocation). The results are represented by the "0.00001_epsilon" graph in Figure 6.10, where $\epsilon \in [0, 0.00001]$. The effect is very limited, and some instances were even slower than the respective unmodified "ring to ring" problem instances. This can be explained by the circumstance that the symmetry is broken, and that the solver thus needs more time in instances where it previously detected symmetry and took advantage of it.

We also observe this in an experiment where node placement preferences are modified by adding an additional uniform random number $k \in [0, 9]$ to node resource weights. The results are shown as "9_nodes_0.0001" graph in Figure 6.10. While showing a speedup in many cases, this experiment crosses the 12h threshold already at 24 VNodes. We therefore conclude that adding uniform random values to weights is not a effective approach to reduce runtime.

6.4 Clustering approximations

In Chapter 5, we confirmed the intuition that the substrate size impacts on runtimes, even though almost all runtime experiments have some jitter. This nonetheless motivates another approximation approach: clustering. In the following, we do not attempt to find a perfect clustering function, but rather use one as an exemple. The rationale for this is that there are no universal optimization criteria [79] for clustering. Moreover, its effect is likely to correlate with the substrate topology and chosen objective function.

6.4.1 Clustering algorithm

Our clustering algorithm is based on k-clustering and works as follows: We precompute all pairwise SNode distances using the Floyd-Warshall algorithm. We then successively select x representants for x clusters. We first select the two SNodes with the highest mutual distance, and then iteratively select the SNode based on its sum of distances to already selected ones. Remaining SNodes are attributed to the closest representant. Ties are broken towards the representant with the lowest database id (see Chapter 8).

This clustering procedure is repeated recursively until the resulting clusters contain at most y SNodes. In our experiments, we selected $x = 2$ and $y = 10$.

Clusters are realized by using feature type (compare Chapter 4) cluster ids to the substrate. VNet components receive one cluster ID which needs to be satisfied. Some variables are only expressed for substrate elements of the respective cluster.

6.4.2 Cluster selection

A good choice of clusters for mapping attempts can yield runtime advantages, while a bad choice likely produces overhead. Cluster selection therefore is of relevance. We select clusters based on an approximation verifying whether node resources would suffice for each cluster. A cluster is only selected if it can support all requested VNode resources. Given that our main objective function produces compact placements, we hence prioritize clusters according the following order:

1. If a cluster features the exact amount of required node resources, select it.
2. Choose the smallest cluster with respect to the number of SNodes.
3. Choose the cluster with the lowest amount of residual node resources.

6.4.3 Iterative placement

Using the clustering approximation, we now consider a series of experiments, where we fill a substrate (with respect to VNodes) by successively adding VNets of different size and topologies. We compare the clustering approach to three other approaches:

Unmodified: The unmodified problem serves as reference point.

No migration: Migrations are disabled by adding placement fixing constraints to the MIP. Our current clustering implementation disables migration. We therefore use the 'No migration' variant to estimate the effect of disallowing reallocation of resources.

DeltaGAP: DeltaGAP is the approach described in Subsection 6.2.4, which does support migration and did not show bogus infeasibility results. Δt was chosen as 5 times the time required to find the first solution.

The substrate used is extracted from rocketfuel topology files [117] and is shown in Figure 6.11. As the randomized nature of our experiments requires repetitive runs, we simulated at least 90 times. We therefore chose a relatively small substrate in order to run the unmodified reference experiment without runtime issues. Moreover, we chose limited link resources to increase the relevance of VNode placement[2]. We iteratively request mapping of VNets of $3-15$ VNodes (uniform at random, except last VNet). The VNet's (uniform random selection) topology are either a line, a star, a ring, a binary tree, or a ring with additional links (uniform at random, 10% propability per link). Binary trees were build by traversing the partially created tree, choosing edges at random and adding new VNode when attempting to traverse a non-existing edge. VNets are accepted greedily whenever they can be supported.

[2]Mind that VNodes use the same amount of resources independent of their location, whereas VLinks are making a difference.

Consider a scenario in which the provider aims to host as many VNodes as possible (and where migration is sufficiently inexpensive). We thus measure quality by the number of hosted VNodes. In order to parallelize we ran experiments with only 1 thread and 900MB RAM per CPLEX instance.

Figure 6.12 shows the results of our testruns. Mind that the 3n_1l scenario does not support rings and supports only limited sizes of other topologies due to its resource constraints. This is why even Unmodified was unable to accomodate all VNets, but holds a significant advantage over approaches not supporting migration. The same holds to a lesser extent for 3n_2l. In experiments with 3n_5l, all approaches are able to embed almost all VNet requests, and the migration advantage was almost gone (with respect to the chosen metric). The individual testruns are aborted after a runtime of 30 minutes. The aborted instances are counted as mapping failure and only sucessful mappings are considered in the statistics. This constitutes a runtime bias in favour of the unmodified version.

Nonetheless, all approximations show runtime advantages. DeltaGAP is effectively capable of capping the runtime without apparent adverse effect on the number of hosted VNodes. This runtime reduction increases with the number of viable mapping solutions (3n_5l). The non-migrating approaches yield results of comparable quality. A slight improvement can be observed with custering. This likely relates to the circumstance that our objective function creates compact VNet embeddings. Our clustering approach groups them together, thus reducing the likelyhood of early placements on bottlenecks. The gap in runtime increases for the NoMigration (compared to the clustering) variant as substrate link resource are increased.

While there are a large number of influence factors (objective functions, topologies, capacities, requested resources, ...), our initial results identify certain trends. DeltaGAP is a useful approach to cap the most excessive runtimes at affordable to no costs. Disabling migration is an option when links are not bottlenecks, and clustering is another viable approach for this scenario.

6.5 Summary

In this chapter, we examine and evaluate different approaches to approximate VNet embeddings. Assessing the situation, we find that not only network sizes but also topology matters. While lines and rings with the same number of network elements yield different runtimes, star topologies are simple to handle.

Upon closer inspection of solver behavior, we find that solvers spend significant time on proving optimality while often finding good to optimal solutions at early stages. Both setting a time limit and allowing for slack in optimality reduces runtime, but fails to yield solutions for all instances. Monitoring how the GAP develops over time yields visible runtime advantages and finds solutions in all experiment instances at

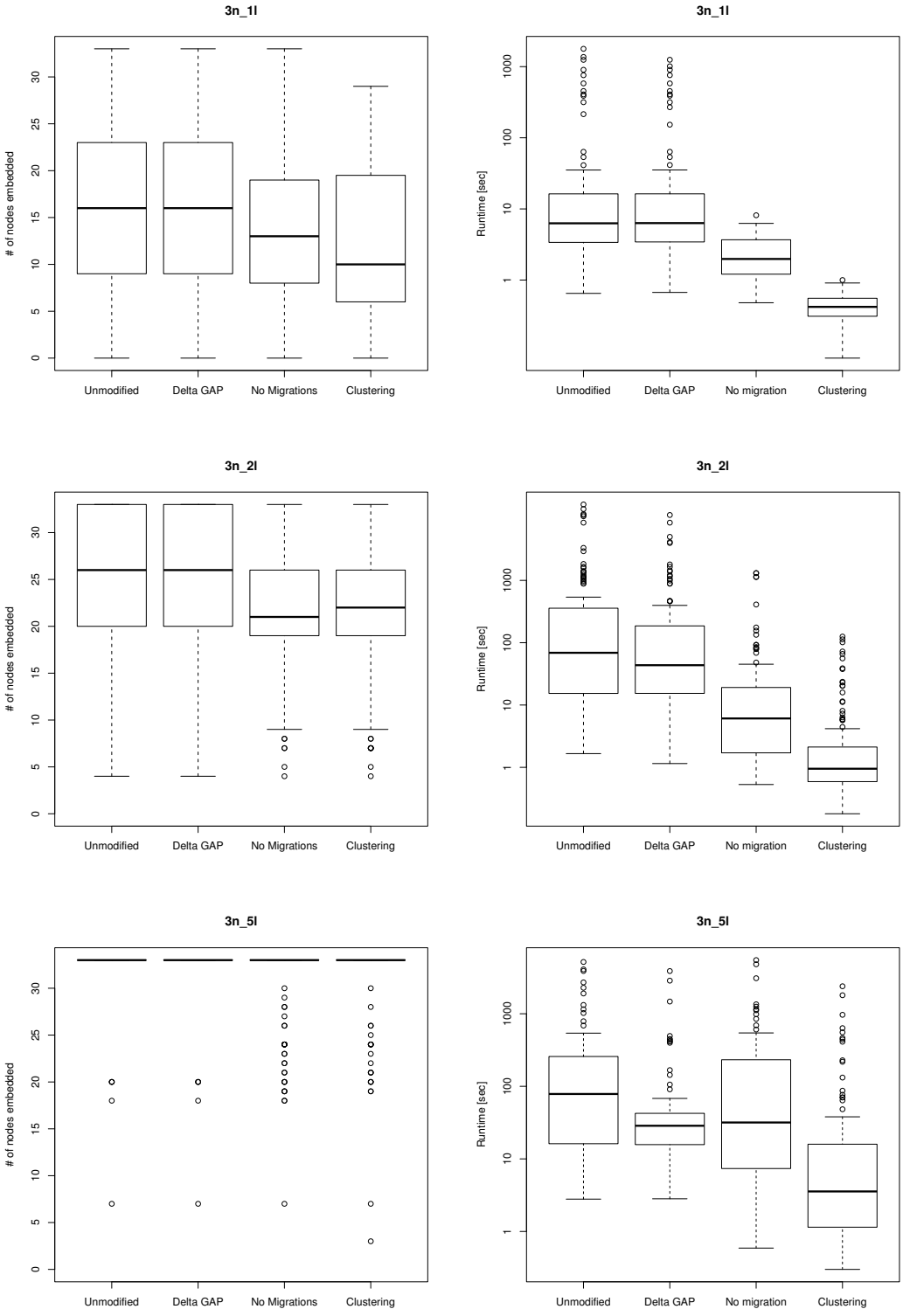


Figure 6.12: **Results for different parameter sets**

reasonable costs, making it an attractive approach. Interestingly, we find that while the solver is taking different solving paths when informed about a time limit, it is not always quicker (or quick enough) in finding the initial solution.

Formulation changes can in general have (non-obvious) effects on solving time [80]. However, our approaches to change the problem formulation had limited success. While the removal of objective functions does have a positive runtime effect, it does so at the expense of an arbitrary quality loss. The attempt to break symmetry by adding uniform random weigths did not succeed. Indeed, it can cause adverse effects.

The benefit of migration is highest when links are bottlenecks. If this is not the case, using iterative placements and heuristics is attractive. In scenarios with severe link resource bottlenecks or when overallocation is to be avoided, the DeltaGAP approach proves useful. But it is otherwise outperformed by clustering in our scenario. Note that the removal of migration in clustering does not need to be absolute. It may be facilitated within the scope of the cluster.

In summary, we recommend clustering approaches harmonizing with chosen scenarios when (global) migration is not crucial. Otherwise, the GAP parameter relaxation and DeltaGAP are powerful mechanisms.

Chapter 7

Migration

One question is how to use the potential advantage of migrations. Clearly, one use lies in the possibility to host additional VNets whose placement would otherwise fail due to misallocations on a substrate bottleneck. In previous chapters, we've assumed a scenario in which a customer - by default - is able to formulate the requirements for his virtual infrastructure based on insight into its expected use. Likewise, we've assumed providers to be capable of estimating load inferred on their substrate. In their own optimization efforts, this allowed them to derive and express their priorities, e.g., in terms of placement weights.

Another use lies in the dynamization of embeddings. Migrations may allow for adaptive embeddings as a function of load fluctuations in the VNet or the substrate. In this Chapter, we now shift the focus to a scenario, in which (e.g., by introduction of a truly novel service) this load is no longer forseeable. As a consequence, the parties are confronted with the question of which requirement formulations (on behalf of the customer), or which placement preferences (on behalf of the provider) would indeed serve their mid- or long-term efficiency goals. At this point, the solving of immediate problems, optimizing only for a single point in time is no longer satisfactory. Instead, qualitative guarantees over a longer period in time become attractive.

In this context, consider a client-server scenario, in which client terminals access a server and thereby produce access costs. As a first step, to provide initial insight into the topic, we consider a simplified scenario where these access costs correlate with latency between server and terminal. In effect, we focus on one single VNet and how it may change over time. The server may migrate 'closer' to specific access points via which the terminals connect to the VNet, producing migration cost. The obvious goal now is to minimize both access and migration costs. Still, the question is: When should a resource be migrated and when not?

Known regular access patterns, e.g, derived out of logs, may allow to use offline algorithms to a certain extent. As an example, MIPs, such as the one presented in Chapter 5 could serve to pre-calculate a number of embeddings for characteristic load constellations in a VNet or substrate. Access patterns, however may not be available at the deployment of a new service, and unforseen events may always create exceptions. Consequently, the dynamics of the scenario and the lack of knowledge about future events eliminate the basis for rational selection of even pre-calculated

embeddings. These situations require so-called *competitive online algorithms*, i.e., algorithms which do not rely on any knowledge about the future but are able to guarantee performance boundaries with respect to optimal offline algorithms. Such performance guarantees are then commonly given in form of a so-called *competitive ratio*, and form the focus of our evaluation interest in this part of the thesis.

This chapter first explains the specific scenario we adress, puts it into context with respect to our architecture, and clarifies the cost factors assumed. It then presents two online single-server algorithm variants Mix and Cen, and derives their competitiveness by worst case analysis. It follows up with an optimal offline algorithm and sketches an extension to a multi-server algorithm. Additionaly, it evaluates Mix and Cen empirically in simulations. It concludes with a short discussion of the results.

7.1 Scenario

As pointed out in previous chapters, one of the possible advantages of virtualization lies in its ease of dynamic resource management. Embeddings may be optimized by reallocation (migration) of resources on any level of abstraction by any role in our role model.

Naturally, providers are limited in the set of possible reoptimizations by the requirements of their customers both in terms of resource properties and requested topologies. Changing, e.g., the topology of a VNet as a provider is likely to require the customer's permission and additional interfaces for coordination. In other cases, the provider may have near full freedom to apply our algorithms. As an example, consider a VNet where an 'access' segment requested as one single link to which all access points and (front end) servers interconnect. In this case, the provider may apply the algorithms to this segment and re-embed servers on a large scale without noticeable topological effect on behalf of his customer. On the VNO side however, even topology changes create no conceptual problem, as long as he receives notification of when his requested changes are effectuated.

Note that in this chapter, we relax the embedding transparency we otherwise assume by default in the previous parts of this thesis. We take this measure of simplification in order to focus on the principal question addressed, and in order to render the algorithm's operations more intuitive to grasp. We however describe in the following what parameters would be based on in the context of our different roles, if embedding transparency were in place.

The core elements in our scenario consist of a set of migratable virtual servers hosting services accessed by terminals via access points of fixed locations they are connected to. The cost metrics and scopes depend on the respective scenarios. Consider the following examples:

PIP environment: Communication between access points (or terminals for that matter) and the server creates traffic load on substrate links. Although service level cost factors remain abstracted as opportunistic costs w.r.t. his customer, the topological distance creates access costs in terms of link usage. Referring to Section 5.2, migration costs consist of C_{mgmt}, C_{reconfig}, C_{transit}, and $C_{\text{adaptation}}$. Migration destinations are the PIP's substrate hosts.

VNP environment: If hosting tariffs are dynamic with respect to actual resource usage, or if the VNP role is taken in the context of a substructured provider, the PIP's access costs may translate into monetary costs on the VNP's side. For VNPs, both service level and resource usage costs can be abstracted by contractual relationships. We deem it not unrealistic that a PIP will simplify tariffs for migration request by a package price derived from his own (e.g., worst case) cost, but the VNP may additionally incur migration costs if he migrates across providers. C_{contract} can be imposed by the origin, or C_{transit} be factured (directly, or via the original PIP) by transit PIPs. Migration destinations may be either named in terms of PIPs, new latency requirements, or by referencing to positioned access points.

VNO/SP environment: Aside from the possibility of usage-based hosting tariffs, the (SP domain) service may, e.g., incur access costs in terms of latency the VNO has to account for (which may also loosely relate to hosting costs on the provider side). In the following, we adopt this as prime motivator for migrations. Assuming that actual PIP or substrate placements are abstracted from this role, migration costs present themselves as configuration transition costs, which may be weighted against the access costs. Migration destinations may be access points (position requirements), or chosen transparently by providers as a function of link requirements (e.g., latency constraints).

While especially PIP level cost factors can vary, the knowledge and capability to find good approximations for projected costs are fundamental to every business. We therefore assume that the entity applying our online algorithms is capable of finding reasonable generic approximations of migration costs and, as a first step, simplify to the following cost and request model.

7.1.1 Model

In the following we consider a VNO scenario, in which hosting and migration costs are prenegotiated. Migration costs are assumed to relate to bulk data transfer over bandwidth reserved for this purpose. While bandwidth capacities and substrate links themselves are not known to the VNO, we relax embedding transparency in this chapter and consider a substrate topology to clarify cost relations. Since our approach does not rely on specific cost factors, this happens at no loss of generality.

Formally, we consider a substrate network $G = (V, E)$ managed by one or multiple substrate providers (PIP). Each of n substrate nodes $v_i \in V$ has a certain strength $\omega(v)$ associated with it (number of CPU cores, memory size, bus speed, etc.). Similarly, each link $e = (u, v) \in E$, with $u, v \in V$, has certain properties, e.g., it is characterized by a latency $\lambda(e)$, and it offers the migration bandwidth capacity $\omega(e)$[1].

In addition to the substrate network, there is a set T of external machines (the mobile thin clients or simply *terminals*) that access G by issuing requests to a virtualized service hosted on a set of up to k virtual servers by G. We assume that there is at most one server per substrate node for a given service. Except for Section 7.4, we focus on a single-service, single-server scenario only.

In order for the machines in T to access the services $\mathcal{S}$, a fixed subset of nodes $A \subseteq V$ serve as *Access Points* where machines in T can connect to G. Due to the movement of machines in T, the access points can change frequently, which may trigger the migration algorithm. We define σ_t to be the multi-set of requests at time t where each element is a tuple $(a \in A, s \in \mathcal{S})$ specifying the access point and the requested service s. (For ease of notation, when clear from the context, we sometimes simply write $v \in \sigma_t$ to denote the multi-set of access points used by the different requests, and use the terms service and server as synonymes). We assume that requests are routed to the closest server.

Service access cost Cost_{acc} at time t are given by the sum of all request path $p_r : a \rightsquigarrow s$ latencies $\text{delay}(r_t) = \sum_{e \in p_r} \lambda(e)$, server load induced by $\eta(v, t)$ answered requests $\text{load}(r_t) = h(\omega(v), \eta(v, t))$ over all requests r_t:

$$\text{Cost}_{\text{acc}} = \sum_{r_t \in \sigma_t} f\left(\text{delay}(r_t), \text{load}(r_t)\right)$$

Migration costs Cost_{mig} in terms of opportunistic outage and bulk data transfer depend on the available bandwidth $\omega(p_m)$ on the migration path $p_m : src \rightsquigarrow dst$, the size $\text{size}(s)$ of server s, and *transit costs* correlating with the number m of transit PIPs on the path. For some function g, where the migration cost is zero if $src = dst$:

$$\text{Cost}_{\text{mig}}(t) = \sum_{s \in S} g(\omega(p_m), k, \text{size}(s))$$

If up to k redundant servers are hosted, Cost_{run} may be considered. Each server can assume three different states: *not in use*, *inactive*, and *active*. If a server is not in use, there are no costs. An inactive server comes at a certain cost C_i per time: this cost includes storing the application software (e.g., a game) plus certain maintenance costs. The running cost of an active server C_a is larger, as it also includes CPU costs, maintaining state in the RAM, or bandwidth costs. In order to

[1] As a simplification, we consider only the dedicated migration bandwidth, and consider VLink bandwidth included in hosting costs

startup a server which is not in use, a fixed creation cost c is assumed. For instance, this cost captures the installation of the Linux box and the template (copy if already on disk or download from an NFS share), configuration of the template (e.g., setting up IP addresses manually or via DHCP), starting the server etc. Finally, we assume that the cost of changing from inactive to active state is negligible.

Our model so far lacks one additional ingredient: *request dynamics* (e.g., due to time-zone effects or due to user mobility). One approach is to assume arbitrary request sets σ_t, where σ_t is completely independent of σ_{t-1}. However, for certain applications it may be more realistic to assume that the mobile nodes move "slowly" between the access points. Note that while users typically travel between different cities or countries at a limited speed, these geographical movements may not translate to the topology of the substrate network. Thus, rather than modeling the users to travel along the links of G, we consider on/off models where a user appears at some access point $a_1 \in A$ at time t, remains there for a certain period Δt, before moving to another arbitrary node $a_2 \in A$ at time $t + \Delta t$.

One may assume that Δt is exponentially distributed. However, in our formal analysis we assume a worst-case perspective and consider arbitrary distributions for Δt. Often, it is reasonable to assume some form of correlation between the individual terminal movements. For example, in an urban area, workers commute downtown in the morning and return to suburbs in the evening. Or in a planetary-scale substrate network, time zone aspects have to be taken into account in the sense that during a day, first many requests will originate from Asian countries, followed by an active period in Europe and finally America. However, as it is rather hard to describe and characterize such movement accurately we, in the formal part, perform a worst case analysis (w.r.t. latency) that does not use any statistical assumptions.

To what extent the system can benefit from virtual network support and migration depends on several factors, e.g., how frequently the thin clients change the access points. Given rapid changes it may be best to place the server in the middle of the network and leave it there. On the other hand, if the changes are slower or can be predicted, it can be worthwhile to migrate the server to follow the mobility pattern. This constitutes the trade-off motivating this chapter.

7.2 Competitive Online Algorithms

In this section, we present two variants of our online algorithm for single server scenarios: Mix chooses randomized migration destinations, whereas Cen chooses a gravity center as destination. Note that due to its deterministic nature, Cen faces a stronger adversary than Mix. While this yields weaker bounds for competitiveness, we expect it to outperform Mix in real scenarios.

7.2.1 Randomized Online Algorithm

The basic idea of Mix is to strike a balance between the request latency cost $\text{Cost}_{\text{acc}}^{\text{Mix}}$ and the migration cost $\text{Cost}_{\text{mig}}^{\text{Mix}}$ it incurs, and to continuously move closer to a possible optimal position. The intuition is that after a small number of migrations only, either Mix is at the optimal position, or an optimal offline algorithm Opt must have migrated as well during this time period. Either way, Opt cannot incur much smaller costs than Mix. In other words, by using Mix for moving to good locations in the network, a possible offline algorithm that migrates less frequently cannot have much lower access costs than Mix; on the other hand, an offline strategy with frequent migrations will have similar costs to Cost_{mig}. In the following, we start with Mix for a basic single PIP scenario, and then extend it to Mix_k for multiple PIPs.

7.2.1.1 Single Provider 'Mix'

Let us first consider a scenario with constant bandwidth capacities, i.e., $\omega(e) = \omega \ \ \forall e \in E$ and let $\beta = \text{size}(s)/\omega$ be the corresponding migration cost, $1 > \phi > 0$ a threshold and c a weighting factor.

The algorithm Mix divides time into *epochs*. In each epoch Mix monitors, for each node v, the cost of serving all requests from this epoch by a server kept at v. We denote this counter by $\text{C}(v)$. Mix keeps the server at a single node w till $\text{C}(w)$ reaches $c \cdot \beta$. In this case, Mix migrates the server to a node u chosen uniformly at random among nodes with the property $\text{C}(u) < \phi \cdot c \cdot \beta$. If there is no such node, Mix does not migrate the server, and the epoch ends in that round; the next epoch starts in the next round and the counters $\text{C}(v)$ are reset to zero.

Lemma 7.2.1. Mix *is* $O(\log n)$*-competitive in networks with constant bandwidth.*

Proof. Fix any epoch $\mathcal{E}$ and let β denote the migration cost. If Opt migrates the server within $\mathcal{E}$, it pays β. Otherwise it keeps it at a single node paying the value of the corresponding counter at the end of $\mathcal{E}$. By the construction of Mix, this value is at least $\phi \cdot c \cdot \beta$, and thus in either case $\text{Opt}(\mathcal{E}) \geq \beta$.

The migrations performed by Mix partition $\mathcal{E}$ into several *phases*. According to our migration strategy, the access cost of Mix in each phase is at most $2c \cdot \beta$. Below, we show that the expected number of migrations within one epoch is at most H_n, where H_n is the n-th harmonic number. The number of phases is then $H_n + 1$, and hence $\text{Mix}(\mathcal{E}) \leq \beta \cdot H_n + \beta \cdot (H_n + 1) = \beta \cdot O(\log n)$. This yields the competitiveness of Mix.

Let $\{v_i\}_{i=1}^n$ be the sequence of nodes in order their counters reach the value β (with ties broken arbitrarily). Assume that in a phase MIX keeps the server at node v_i and let T_i be the expected number of server migrations until the end of $\mathcal{E}$. If $i = n$, then the current phase is the last one in $\mathcal{E}$, and thus $T_n = 0$. Otherwise $i < n$, and at the end of this phase MIX chooses a next place for the server uniformly at random from the set $v_j : i < j \leq n$. Hence, we obtain the recursive formula: $T_i = 1 + \sum_{j=i+1}^{n} (\frac{1}{n-i} T_j)$

By a straightforward induction, one may show that $T_i = H_{n-i}$. Thus, if MIX starts $\mathcal{E}$ with server at node v_k, the expected number of migrations in $\mathcal{E}$ is $H_{n-k} \leq H_n$. □

Note that the analysis does not rely on access costs being measured as the number of hops. Rather, the analysis (and hence also the result) is applicable to any metric which ensures that counters increase monotonically over time, i.e., with additional requests.

Theorem 7.2.2. MIX *is* $O(\mu \cdot \log n)$*-competitive in general networks, where* $\mu = \max_{e,e' \in E} \omega(e)/\omega(e')$.

Proof. In networks with general bandwidths, β is an approximation of migration costs. For instance, MIX can be adopted in such a way that it migrates when the counter of the current location v reaches $\text{size}(s)/\min_e \omega(e)$, that is, when $\text{C}(v) \geq \text{size}(s)/\min_e \omega(e)$. Thus, the cost of the optimal algorithm in each epoch is at least $\text{size}(s)/\max_e \omega(e)$, while the cost of MIX is at most $2 \cdot \text{size}(s)/\min_e \omega(e)$. Therefore, by the same arguments as in the proof of Lemma 7.2.1, we immediately obtain the above Theorem. □

7.2.1.2 Multiple Providers 'Mix$_k$'

The flexibility offered by network virtualization is not limited to a single PIP. In the following, we extend our model to multiple provider scenarios. In particular, we assume that migrating a server across a PIP boundary entails a fixed "roaming" cost π for each transit.

In the following, we present how the randomized algorithm MIX can be extended to multi-PIP scenarios. We consider k PIPs, migration inside a PIP costs β, access costs are the number of hops, and migrating across providers costs π per crossed PIP boundary. We concentrate on the more realistic case where $\pi \geq \beta$.[2]

It is sometimes useful to think of the *PIP graph*, the graph where all the nodes of one PIP form one vertex and two PIPs are connected if there is a connection between nodes of the respective PIPs. In particular, we will refer to the *diameter of the PIP*

[2] If the total migration cost (over multiple providers) is in the order of β, our single PIP algorithms could be applied without taking into account transit costs.

graph, the largest number of PIPs to be traversed on a shortest migration path, by Δ.

Algorithm MIX$_k$ generalizes MIX by moving the server to one of the PIPs having lower costs.

The algorithm MIX$_k$ divides time into three types of *epochs*: *huge epochs* which consist of one or several *large epochs* which in turn consist of $\lceil \pi/\beta \rceil$ *small epochs*. For each node u, we use two counters $\mathrm{C}(u)$ and $\mathrm{C}_L(u)$ to count the access cost during a small and a large epoch, respectively. At the beginning of a small epoch, all nodes u are *active* (i.e., $\mathrm{C}(u) < \phi \cdot c \cdot \beta$) ; similarly, at the beginning of a huge epoch, we say that all PIPs are *active* (i.e., they own active nodes). During a small epoch, the server is migrated within a single PIP only, until there is no node u left with access costs smaller than $\phi \cdot c \cdot \beta$: MIX$_k$ monitors, for each node u, the cost of serving all requests from this small epoch by a server kept at u; MIX$_k$ keeps the server at a single node u till $\mathrm{C}(u)$ reaches $\phi \cdot c \cdot \beta$. When this happens, MIX$_k$ migrates the server to a node v chosen uniformly at random among nodes of the current PIP with the property $\mathrm{C}(v) < \phi \cdot c \cdot \beta$. If there is no such node, MIX$_k$ does not migrate the server, and the small epoch ends in that round; the next epoch starts in the next round and the counters $\mathrm{C}(u)$ are reset to zero.

After $\lceil \pi/\beta \rceil$ small epochs a large epoch ends. Then MIX$_k$ determines the set of PIPs that contain at least one node v for which $\mathrm{C}_L(v) < \phi' \cdot \pi$; all other PIPs become inactive for the remainder of the current huge epoch. If there are active PIPs left, MIX$_k$ chooses an active PIP uniformly at random and migrates to an arbitrary node of that PIP; otherwise the server stays where it is, all counters are reset, and a new huge epoch begins.

We can derive the following competitive ratio on MIX$_k$'s performance.

Theorem 7.2.3. MIX$_k$ *is* $O(\log k \cdot (\log n_1 + \Delta))$*-competitive in networks with constant bandwidth and* k *PIPs, where* n_1 *is the size of the largest PIP, and* Δ *is the "diameter of the PIP graph".*

Proof. From Lemma 7.2.1, we know that during a small epoch, MIX$_k$ accumulates a cost of at most $O(\beta \log n_1)$: There is at most a logarithmic number of migrations and the access costs per phase is at most β. Recall, that a large epoch consists of at most $\lceil \pi/\beta \rceil$ many small epochs, and subsequently, a remaining active PIP is chosen uniformly at random. Thus, similarly to Lemma 7.2.1, it holds that there are at most $O(\log k)$ many large epochs, yielding a total access cost of $O(\log k \cdot \log n_1 \cdot \pi/\beta \cdot \beta) = O(\pi \log k \cdot \log n_1)$. The migration costs within PIPs are of the same order. The transit costs to move the server between PIPs amounts to at most $O(\Delta \pi \log k)$. Thus, the overall cost of MIX$_k$ per huge epoch is in the order of $O(\pi \log k \cdot (\log n_1 + \Delta))$. On the other hand, an optimal offline algorithm must have had costs of at least $\phi' \cdot \pi$ as

well during this huge epoch: if the optimal algorithm migrates between PIPs, the claim follows trivially. Otherwise, the optimal offline algorithm is located at a single PIP during the entire huge epoch; by the construction of Mix_k, there must exist a large epoch in which the optimal offline algorithm incurred a cost of at least $\Omega(\pi)$: per small epoch the (access or migration) costs are at least β, and there are $\lceil \pi/\beta \rceil$ many small epochs in a large epoch. The claim follows. □

Note that the proof of Theorem 7.2.3 is overly pessimistic, as it assumes several large migration distances that the optimal offline algorithm can avoid. We believe that Mix_k performs better, also in the worst-case, a conjecture that is also manifested by our experiments.

7.2.2 Deterministic Online Algorithm 'Cen'

While for the analysis of Mix, we assume an oblivious adversary which cannot be adaptive with respect to the random choices made by the online algorithm, we now focus on deterministic algorithms Cen. As we will see, a logarithmic competitive ratio can also be achieved. We will first assume a weighting factor c, $\omega(e) = \omega \ \ \forall e \in E$ and $\beta = \text{size}(s)/\omega$. Again, we start with Cen for a single PIP scenario, and then extend it to a multi-PIP Cen_k.

7.2.2.1 Single Provider 'Cen'

Cen divides time into *epochs* consisting of one or multiple *phases* between which Cen migrates. Again, we have counters $\text{C}(v)$ for each node v that are set to zero at the beginning of an epoch. These counters accumulate the access costs of an epoch if the server was permanently located at v. Assume a factor $1/2 > \phi > 0$ and a threshold $\tau = \phi \cdot c \cdot \beta$. In the context of Cen, we call all nodes v for which at time t, $\text{C}(v) < \tau$, *active nodes* at time t. Assume that algorithm Cen is currently at some node v. Cen remains at this node until it accumulated there access costs of $c \cdot \beta$. Then, a new phase starts, and Cen computes the *gravity center* w, i.e., the "center" of the currently active nodes. Formally, let $d()$ denote the shortest path metric on the weighted network graph G, and a cost function $f : f(a) \geq f(b) \Leftrightarrow a \geq b$. The gravity center of a subset $V' \subseteq V$ of nodes is defined as the (not necessarily unique) node $\mathcal{G}(V') = \arg\min_{v \in V'} \sum_{u \in V'} f(d(u,v))$. (Ties are broken arbitrarily.) Cen migrates to w and a new phase starts. If there is no active node left, the epoch ends.

In order to study the competitive ratio of Cen, we exploit the fact that a request always increases the counter of several nodes besides the gravity center (namely: a constant fraction) by at least a certain value (again, a constant fraction) as well.

Lemma 7.2.4. *Let $1 > \lambda_1, \lambda_2 > 0$, $c = \frac{\lambda_2}{1-\lambda_2}$, and $\lambda_1 \geq \frac{1-c}{3+c}$. Fix any active set V'. Let r be an arbitrary requesting node (at some step). Assume the counter at the gravity center $\mathcal{G}(V')$ increased by $f(F)$ because of this request. Then there are at least $\lambda_2 \cdot |V'|$ nodes from V' whose counters increased at least by $f(\lambda_1 \cdot F)$.*

Proof. Assume the contrary. It means that there are at least $(1-\lambda_2) \cdot |V'|$ nodes from V' whose counter increase is smaller than $f(\lambda_1 \cdot F)$. Denote this set by V''. By construction of $f()$), nodes in V'' have a distance to the request smaller than $\lambda_1 \cdot F$. We know that the distance between the request and the center is $d(\mathcal{G}(V'), r) = F$, and $\forall u \in V''$, $d(u, r) < \lambda_1 \cdot F$. Therefore, $\forall u, v \in V''$, $d(u, v) < 2\lambda_1 \cdot F$: the diameter of the set V'' is relatively small.

Now let ξ be any node of V''. We show that ξ would be a better candidate for the gravity center than $\mathcal{G}(V')$ is. Given that $d(\mathcal{G}(V'), r) = F$, that $d(u, r) \leq \lambda_1 \cdot F$, and that $d(\xi, r) < \lambda_1 \cdot F$, $d(r, \mathcal{G}(V')) = F$, by using triangle inequalities, we obtain

$$\begin{aligned}
\sum_{u \in V'} d(\mathcal{G}(V'), u) &= \sum_{u \in V''} d(\mathcal{G}(V'), u) + \sum_{u \in V' \setminus V''} d(\mathcal{G}(V'), u) \\
&\geq \sum_{u \in V''} \big[d(\mathcal{G}(V'), r) - d(u, r)\big] + \sum_{u \in V' \setminus V''} d(\mathcal{G}(V'), u) \\
&> (1-\lambda_1) \cdot |V''| \cdot F + \sum_{u \in V' \setminus V''} d(\mathcal{G}(V'), u) \qquad (1) \\
\sum_{u \in V'} d(\xi, u) &= \sum_{u \in V''} d(\xi, u) + \sum_{u \in V' \setminus V''} d(\xi, u) \\
&< 2\lambda_1 \cdot |V''| \cdot F + \sum_{u \in V' \setminus V''} \big[d(\xi, r) + d(r, \mathcal{G}(V')) + d(\mathcal{G}(V'), u)\big] \\
&< 2\lambda_1 \cdot |V''| \cdot F + |V' \setminus V''| \cdot (1+\lambda_1) \cdot F + \sum_{u \in V' \setminus V''} d(\mathcal{G}(V'), u) \qquad (2)
\end{aligned}$$

On the other hand, note that by definition of V'', $|V' \setminus V''| \leq \lambda_2 \cdot |V'| \leq c \cdot |V''|$. The above therefore contradicts that $\mathcal{G}(V')$ is the gravity center of V'. Consider $(1) > (2)$:

$$\begin{aligned}
(1-\lambda_1) \cdot |V''| &\geq 2\lambda_1 \cdot |V''| + |V' \setminus V''| \cdot (1+\lambda_1) \\
&\geq 2\lambda_1 \cdot |V''| + c \cdot |V''| \cdot (1+\lambda_1) \\
1 &\geq 3\lambda_1 + c \cdot (1+\lambda_1) \geq \frac{3 - 3c + 3c + c^2 + c - c^2}{3+c} = 1
\end{aligned}$$

□

From Lemma 7.2.4 it follows that when the counter at the gravity center exceeds a given threshold, the counter of many nodes besides the center must be high as well.

Lemma 7.2.5. *Fix any threshold θ and let $\phi = \frac{\lambda_1 \cdot \lambda_2}{2}$ for $1 > \lambda_1, \lambda_2 > 0$, $c = \frac{\lambda_2}{1-\lambda_2}$ and $\lambda_1 \geq \frac{1-c}{3+c}$. When the counter at the gravity center $\mathcal{G}(V')$ exceeds θ, then there exists $V'' \subseteq V'$, $|V''| \geq \frac{\lambda_2}{2} \cdot |V'|$, such that for all $v \in V''$, the counter at v is at least $\phi \cdot \theta$.*

Proof. Assume the contrary, i.e., there exists $V'' \subseteq V'$, $|V''| > (1 - \frac{\lambda_2}{2}) \cdot |V'|$, such that for all $v \in V''$, the counter at v is smaller than $\phi \cdot \theta$. In consequence $\sum_{v \in V''} \mathrm{C}(v) < |V''| \cdot \phi \cdot \theta \leq |V'| \cdot \phi \cdot \theta$. On the other hand, by Lemma 7.2.4, each time the counter $\mathrm{C}(\mathcal{G}(V'))$ increases by F, at least $\lambda_2 \cdot |V'|$ counters from set V' (and hence at least $\frac{\lambda_2}{2} \cdot |V'|$ counters from set V'', since $|V' \setminus V''| \leq \frac{\lambda_2}{2} \cdot |V'|$) increase by $\lambda_1 \cdot F$. Hence, in this case, the sum of counters from V'' increases at least by $\phi \cdot |V'| \cdot F$. Therefore, when $\mathrm{C}(\mathcal{G}(V')) \geq \theta$, then $\sum_{v \in V''} \mathrm{C}(v) \geq |V'| \cdot \phi \cdot \theta$, which is a contradiction. □

For the competitive ratio, we therefore have the following result.

Theorem 7.2.6. Cen *is $O(\log n)$-competitive in constant bandwidth scenarios.*

Proof. First, we consider the cost of the optimal offline algorithm. If Opt migrates in an epoch, it has costs β. Otherwise, due to the definition of Cen, as there are no active nodes left at the end of an epoch, the access costs of any node is also in the order of $\Omega(\beta)$. Regarding Cen, we know that in each phase, access costs are at most $c \cdot \beta$, and it remains to study the number of phases per epoch. By Lemma 7.2.5, we know that in each phase, the number of active nodes is reduced by a factor at least $2/\lambda_2$ for a constant $1 > \lambda_2 > 0$. Therefore, there are at most $O(\log_{\frac{2}{\lambda_2}} n)$ many phases per epoch, and the claim follows. □

Theorem 7.2.7. Cen *is $O(\mu \cdot \log n)$-competitive in general networks, where $\mu = \max_{e,e' \in E} \omega(e)/\omega(e')$.*

Proof. Again, for networks with general bandwidths, Cen can be adopted in such a way that it migrates when the counter of the current location v reaches $\mathrm{size}(s)/\min_e \omega(e)$, that is, when $\mathrm{C}(v) \geq \mathrm{size}(s)/\min_e \omega(e)$. By the same arguments as above, this adds a factor $\max_{e,e' \in E} \omega(e)/\omega(e')$ to the competitive ratio. □

7.2.2.2 Multiple Providers 'Cen$_k$'

Like Mix, it is possible to extend Cen with inter-provider migrations. Again, we consider k PIPs, intra-PIP migration cost β, a weight factor c, and inter-PIP migration costs of π per crossed PIP.

The algorithm CEN_k divides time into three types of *epochs*: a *huge epoch* consists of multiple *large epochs*, and a large epoch consists of $\lceil \pi/\beta \rceil$ *small epochs*. Again, we use counters $\text{C}(u)$ to accumulate the access costs of a node u during a small epoch; in addition, a counter $\text{C}_L(u)$ is used to accumulate access costs during a large epoch. In the beginning, all PIPs are set to *active*. At the beginning of a small epoch, the $\text{C}(u)$ values are set to zero for all nodes within the current PIP. CEN_k then monitors, for each node u, the cost of serving all requests from this small epoch by a server kept at u. CEN_k leaves the server at a single node u till $\text{C}(u)$ reaches $c \cdot \beta$. In this case, CEN_k migrates the server to a node v which constitutes the center of gravity among the *active* nodes of the current PIP, i.e., the nodes w of the current PIP for which it still holds that $\text{C}(w) < \tau$ for some threshold $\tau = \phi \cdot c \cdot \beta$ and some factor $\phi \leq 1/2$. If there is no active node left within the current PIP, a small epoch ends in that round; the next small epoch starts in the next round. After $\lceil \pi/\beta \rceil$ small epochs, a new large epoch starts, and all nodes u with $\text{C}_L(u) \geq \phi' \cdot \pi$ become inactive with respect to the large epoch. Among all remaining active nodes of the large epoch, CEN_k determines their center of gravity and moves the server to the corresponding PIP, and a new large epoch begins. Otherwise, if there is no PIP left that contains active nodes, the server stays where it is, and a new huge epoch starts.

Theorem 7.2.8. *CEN_k is $O(\log n \cdot (\log n_1 + \Delta))$-competitive in networks with constant bandwidth and k PIPs, where n_1 is the size of the largest PIP, and Δ is the diameter of the PIP graph.*

Proof. First we compute the total cost of CEN_k in a huge epoch. It follows from Theorem 7.2.6 that a large epoch consists of $\lceil \pi/\beta \rceil$ small epochs of $O(\beta \log n_1)$ access costs and at a logarithmic number of migrations amounting to cost $O(\beta \log n_1)$ as well, yielding a total cost per large epoch of $O(\pi \log n_1)$. Now observe that there is at most a logarithmic number of large epochs per huge epoch: CEN_k guarantees that the server is not migrated to another PIP as long as there is a node left in the current PIP with access costs smaller than π; in particular, for the center of gravity u of the current large epoch PIP, $\text{C}_L(u) \geq \pi$, and hence, again by Theorem 7.2.6, a constant fraction of nodes in the entire network must become inactive per large epoch. Summing up over the large epochs and adding the transit cost of at most $O(\Delta \pi \log n)$, the total cost is at most $O(\log n \cdot \pi(\log n_1 + \Delta))$. The cost of the optimal offline algorithm can be analyzed similarly to the proof of Theorem 7.2.3: If the offline algorithm migrates between PIPs during a huge epoch, it has a cost of at least π; otherwise, it has either access or migration costs of at least $\phi \cdot c \cdot \beta$ per small epoch and hence $\Omega(\pi)$ per large epoch, and the claim follows. □

Again, we believe that the actual ratio is better, even in the worst-case, as our analysis is pessimistic.

7.3 Optimal Offline Algorithm 'Opt'

In the competitive analysis of our online algorithms we often argued about a hypothetical optimal offline algorithm to which we compare our costs. However, there was no need to find or describe the offline algorithm explicitly. Still, while the decisions when and where to migrate servers typically needs to be done *online*, i.e., without the knowledge of future requests, there can be situations where it is interesting to study which migration pattern would have been optimal *at hindsight*. For example, if it is known that the requests follow a regular pattern (e.g., a periodic pattern per day or week), it can make sense to compute an optimal migration strategy offline and apply it in the future. Another reason for designing optimal offline algorithms explicitly is that an optimal solution is required to compute the competitive ratio in our simulations.

We therefore present an optimal offline algorithm for our server migration problems. It turns out that offline strategies can be computed for many different scenarios, and we describe a very general algorithm here. Similarly to the online algorithms, offline strategies can be computed efficiently both for intra and inter provider migration.

It exploits the fact that migration exhibits an optimal substructure property: Given that at time t, the server is located at a given node u, then the most cost-efficient migration path that leads to this configuration consists solely of optimal sub-paths. That is, if a cost minimizing path to node u at time t leads over a node v at time $t' < t$, then there cannot be a cheaper migration sub-path that leads to v at time t' than the corresponding sub-path.

Opt essentially fills out a matrix opt[time][node] where $opt[t][v]$ contains the cost of the minimal migration path that leads to a configuration where the server satisfies the requests of time t from node v. Assume that initially, the service is located at node v_0. Thus, initially, $opt[0][u] = Cost_{\text{mig}}(v_0, u) + \left[\sum_{v \in \sigma_0} Cost_{\text{acc}}(v, u)\right]$ as the migration origin is v_0, and as a request needs to travel on the access link from the terminal to v and from there to u (w.l.o.g., we assume that the cost $Cost_{\text{acc}}$ contains the first wireless hop from terminal to substrate network).

For $t > 0$, we find the optimal values $opt[t][u]$ by considering the optimal migration paths to any node v at time $t-1$, and adding the migration cost from v to u. That is, in order to find the optimal cost to arrive at a configuration with server at node u at time t:

$$\min_{v \in V} \left[opt[t-1][v] + Cost_{\text{mig}}(v, u) + \sum_{w \in \sigma_t} Cost_{\text{acc}}(w, u) \right]$$

where we assume that $Cost_{\text{acc}}$ includes the first (wireless) hop of the request from the terminal to the substrate network, and where $Cost_{\text{mig}}(v, v) = 0 \ \ \forall v$.

We have the following runtime result.

Theorem 7.3.1. *The optimal offline migration policy* Opt *can be computed in* $O(n^3 + n^2 \sum_{t \in \Gamma} |\sigma_t|)$ *time, where* Γ *is the set of rounds in which events occur.*

Proof. Note that we can constrain ourselves to optimal offline algorithms where migration will only take place in "active" rounds Γ with at least one request. This is useful in case of sparse sequences with few requests. The $opt[\cdot][\cdot]$-matrix contains $|\Gamma| \cdot n$ entries. In order to compute a matrix entry, we need to consider each node $v \in A$ from which a migration can originate; for each such node, the access cost from all the requests in σ_t need to be computed. Both the shortest access paths and the migration costs can be looked up in a pre-computed table (pre-computation in time at most $O(n^3)$, e.g., by *Floyd-Warshall's algorithm*) and require a constant number of operations only, which implies the claim. □

Note that Opt is not an online algorithm, although $opt[t]$ does not depend on future requests. In order to reconstruct the optimal migration strategy at hindsight, the configuration of minimal cost after the last request is determined, and from there, the optimal path is given by recursively finding the optimal configuration at time $t - 1$ which led to the optimal configuration at time t.

7.4 Multi-Server Extension

As a natural extension, Mix may be extended to a multi-server version OnConf by introduction of a configuration notion. In this section, we consider the interesting case where OnConf is considering configuration transitions rather than migrations.

Definition 7.4.1 (Configuration)**.** *A* configuration γ *describes, for each server, whether it is* not in use*,* inactive*, or* active*. In case of inactive and active servers,* γ *specifies where—i.e., on which node—the server is located.*

Online Algorithm OnConf: OnConf uses a counter $\mathrm{C}(\gamma)$ for each configuration γ. Time is divided into *epochs*. In each epoch OnConf monitors, for each configuration γ, the cost of serving all requests from this epoch by servers kept in configuration γ, including the access costs (latency plus induced load) of the requests, and the server running costs OnConf stores this cost in $\mathrm{C}(\gamma)$. The servers are kept in a given configuration $\widehat{\gamma}$ until $\mathrm{C}(\widehat{\gamma})$ reaches $\tau = \max\{\beta \cdot k/2, (k-1) \cdot c\}$,

where k is the maximal number of servers. In this case, OnConf changes to a configuration $\widehat{\gamma}'$ chosen uniformly at random among configurations with the property $\mathrm{C}(\gamma) < \tau$. If there is no such configuration left, the epoch ends in that round; the next epoch starts in the next round and the counters $\mathrm{C}(\gamma)$ are reset to zero.

During an epoch, there are n possible locations for up to k instantiated servers, $k-1$ of which may be in one of two states (running, inactive). Therefore, there exist $|C| = \sum_{i=1}^{k} \binom{2n}{i} - n = \sum_{i=1}^{k} \prod_{j=0}^{i-1} \frac{2n-j}{i-j} - n \leq k(2n)^k$ many configurations in total. By argumentation analogue to Lemma 7.2.1, OnConf visits at most $H_{|C|}$ (i.e. $O(k \log(kn) \subseteq O(k \log n)$) many configurations. Assuming at least one request and one active server per round, the cost per epoch is at most $2\tau \leq 2k \max\{\beta, c\}$ (this is overly pessimistic). An optimal offline algorithm run at least one server and served at least one (best case) local request per round to sum up $\Delta t \cdot (f(\text{lasthop\_delay}, \text{load}(r_t)) + C_a) \geq f(\text{lasthop\_delay}, \text{load}(r_t)) + C_a =: C_t$ cost. Thus, we have a worst-case performance guarantee: the competitive ratio is at most $O(\max\{\beta, c\}/C_t \cdot k^2 \log n) = O(k^2 \log n)$.

Clearly, OnConf needs to be optimized in many respects. For instance, it can make sense to switch between "close" (with respect to costs) configurations only, or to deterministically switch to the configuration with the lowest counter. However, the main problem of OnConf is different: due to the configuration complexity, the runtime is only acceptable for a small number of servers k.

7.5 Evaluation

In order to complement our formal insights and in order to study the behavior of our algorithms in different environments, we implemented a simulation framework. In the following, we report on some of our results in more detail.

In the following, we first introduce the chosen evaluation scenario and its set-up. We then describe two additional approaches STAT and TIMM we use for comparison and proceed with a single-provider and a multi-provider evaluation.

7.5.1 Scenario

We chose a time zone scenario to evaluate our algorithms and to compare them to the two subsequently described other approaches. Note that geographic and topological distance correlates only to a certain extent in the real world. We therefore chose to follow an on-off model to model terminal mobility rather than moving terminals along the topology.

Time Zones Scenario: This scenario models an access pattern that can result , e.g., from global daytime effects. A *current* zone is chosen round-robin from the set of available zones every k turns ($k = 16$ in our experiments). At each time t, new requests originate from nodes chosen uniformly at random from the respective current zone, until they represent a fraction p of all requests. The remaining request locations are chosen uniformly at random from all nodes. Request durations follow an exponentially distributed sojourn time with parameter λ.

7.5.2 Set-Up

We conducted experiments on connected *Erdös-Rényi* random graphs[3] with 30-130 nodes and an average 1.4 links per node[4]. The graphs were partitioned in 8 zones. Single provider scenarios comprised 8 zones of equal size. Each provider in multi-provider scenario is associated to one of the 8 zones respectively and interconnected to a ring (choosing one random pair of border gateways per connection).

Link bandwidths are chosen at random (either T1 (1.544 Mbit/s) or T2 (6.312 Mbit/s)), and the server size is 2048MB. If not stated otherwise, we assume that $c = 1$, that β equals the server size divided by the average bandwidth, that $\tau = 11/2$, that $x = \pi/\beta = 3$. To provide an incentive for migration, inter-zone links are weighted with a latency of 100 whereas other links bear a unit weight. Experiments were run for 400 turns, and results were averaged over 5 repetitions. All compared algorithms start at the center of the (weighted) graph.

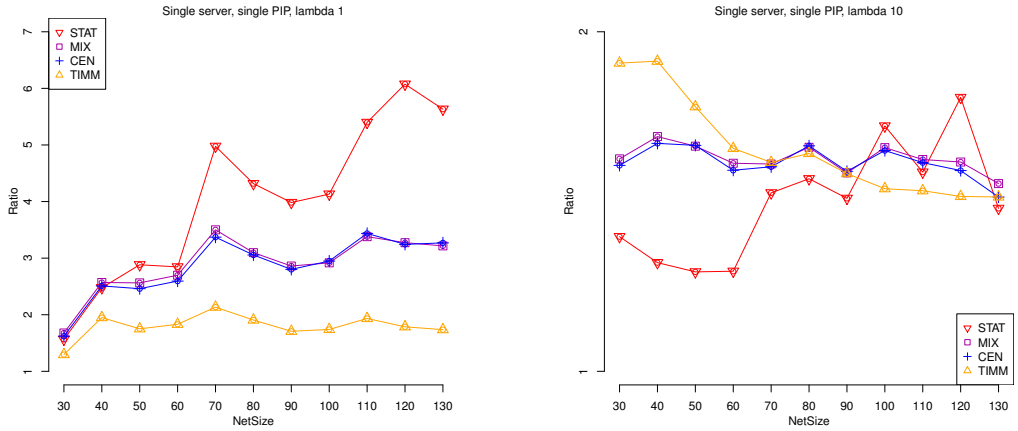
To take into account that larger networks typically come with a larger requests set, we assume that the number of requests per round is one fifth of the network size. Note that the runtime of the optimal offline algorithm and hence the computation of the competitive ratio is expensive in large networks; therefore, the scale of our experiments is typically limited.

7.5.3 Additional algorithms

Additionally, we evaluate Mix and Cen performance in comparison with two canonic approaches to handle uncertainty in deployment of new services. It is possible on one hand to place the server at a presumed optimal position, and to leave it there while gathering initial data on usage patterns. On the other hand, it may be an option to assume usage patterns on a coarse granularity (e.g., assuming patterns to follow a regular time-of-day routine), and to start off, by scheduling migrations based

[3] first creating an E.-R. graph, then repeatedly select two disconnected segments uniform at random to create one uniform random connection between them

[4] by choosing the link probability in relation to the number of nodes n



(a) $\lambda = 1$ (b) $\lambda = 10$

Figure 7.1: **Competitive ratio as a function of network size, averaged over all** $p \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$

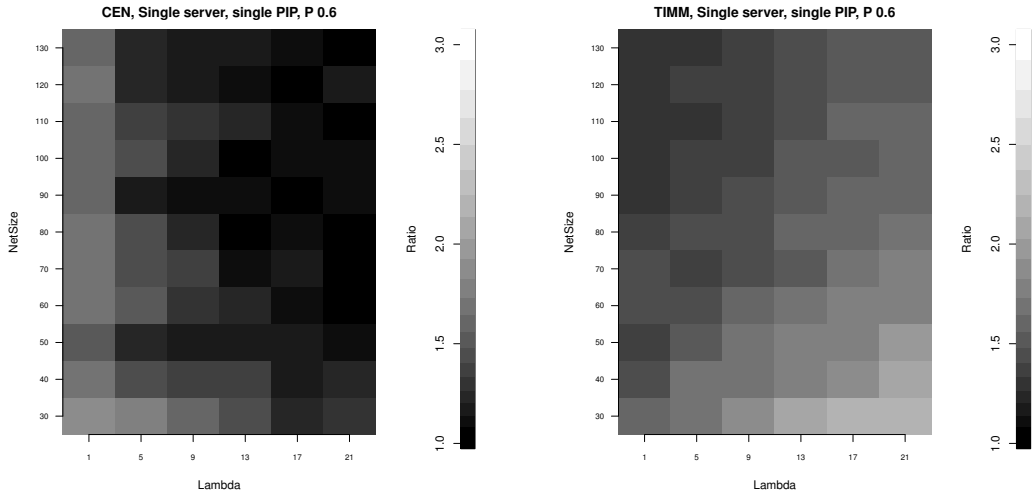
on these assumptions. These two approaches are reflected in the below algorithms STAT and TIMM.

STAT is a STATic placement of the server at the center of the (weighted) topology, which also serves as starting point for all algorithms. STAT does not migrate.

TIMM performs a TIMed Migration every k rounds. It follows the official cycle of changing zones by chosing a destination uniformly at random amongst all nodes of the new zone.

7.5.4 Intra Provider Migration

We begin with an inspection of the competitive ratio as a function of the number of nodes for a very short $\lambda = 1$ (Figure 7.1(a)) and a longer $\lambda = 10$ (Figure 7.1(b)). Note that a low value of $\lambda = 1$ benefits TIMM by introducing only a minimal delay between the switch of the 'current' zone and the real shift in request weight. TIMM therefore performs formidably in such experiments, while CEN and MIX exhibit a moderate drop in competitiveness. This benefit is shifted towards CEN and MIX with $\lambda = 10$ by allowing them better migration amortization after translocation. TIMM on the other hand suffers from the reduced predictability, migrating ahead of the shift in request weight. Moreover, high values of λ and large networks allow OPT to find more economic migration opportunities, resulting in an increased migration cost. This in turn reduces the ratio values for all algorithms considerably.



(a) Cen (b) *TIMM*

Figure 7.2: **Competitive ratio as a function of λ and network size for** $p = 0.6$

Finally, the difference between Cen and Mix proved to be nominal in almost all cases. We therefore omit Mix plots for most of the following experiments. As to λ, we will focus on the challenging value of 1, and introduce plots of other experiments only to highlight specific properties. This not only gives TIMM a better chance for competition, but also yields clearer results due to the reduced ratio range for $\lambda = 10$ in most cases.

The abovementioned features are also visible in Figure 7.3, which shows the results as a function of request correlation p. All algorithms suffer from a full correlation of $p = 1$, though due to different respects. At low p values, economic migration destinations are scarce and not obvious. Cen and Mix therefore imitate STAT and do not migrate below $p = 0.4$. TIMM migrates nonetheless, incurring high costs. At $p \in \{0.4, 0.6\}$, migration destinations are available, and choosing the 'wrong' zone is not yet dire. This yields good ratios for TIMM and Cen/Mix. At $p = 1$ however, TIMM, Cen and Mix suffer from a bad choice (the prior running ahead, the latter delayed by slowly filling counters). As in the previous plots, $\lambda = 10$ allows Cen/Mix to amortize the late migration, while the minimum of TIMM's ratio curve shifts to the left. As to be expected, STAT yields best results with low correlation and small network sizes.

Focusing on λ, Figures 7.2(a) and 7.2(b) highlight the inverse effect on Cen and TIMM. As effect of the limited amount of requests and our method of choosing request origins, TIMM's ratio increase however remains limited. Very high λ values have the effect of 'jumped' zones. If requests of a zone remain active over the whole period of its successor, requests will jump this zone and thereby 'catch up' with

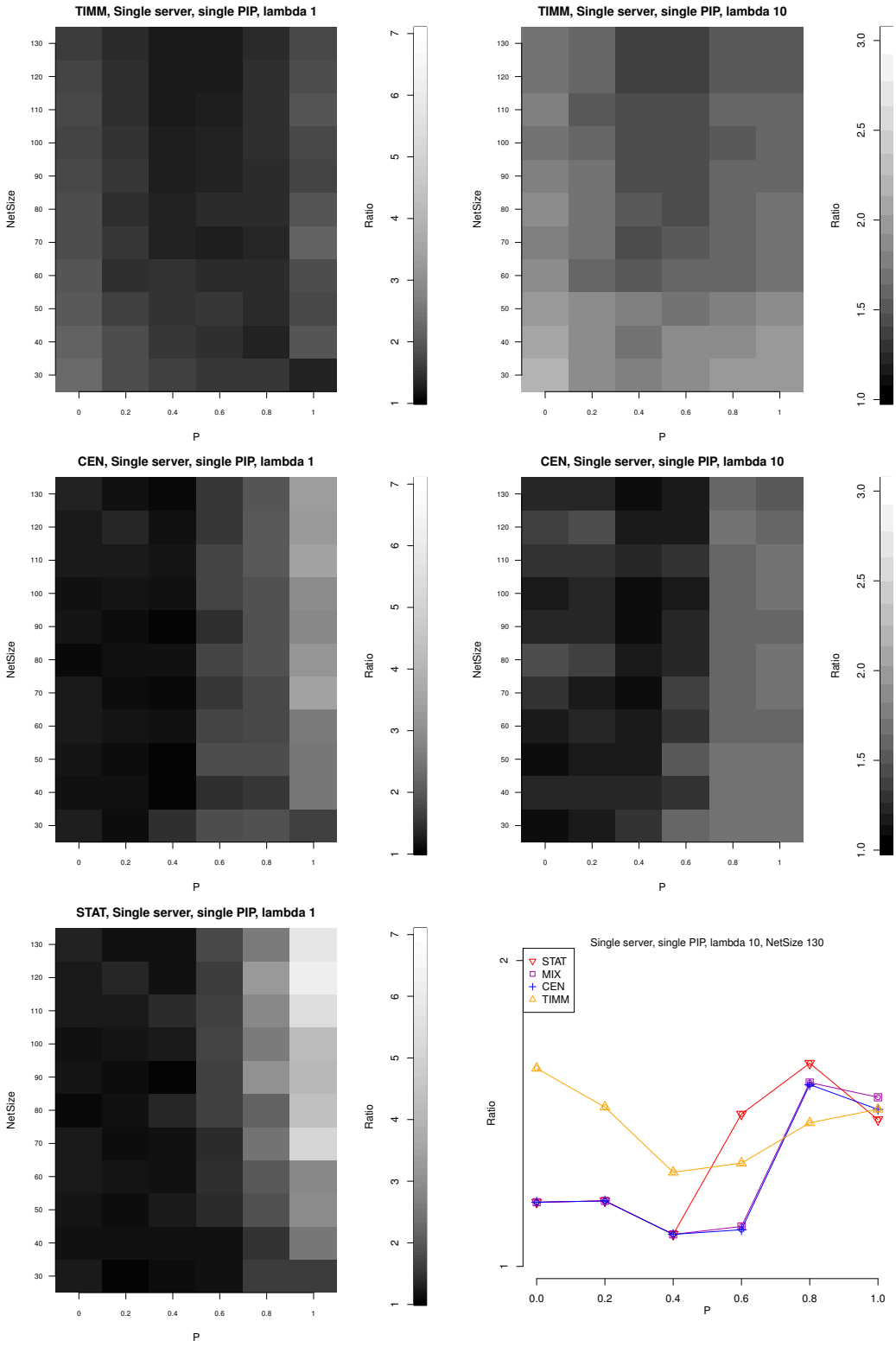


Figure 7.3: **Single provider results for different parameter sets**

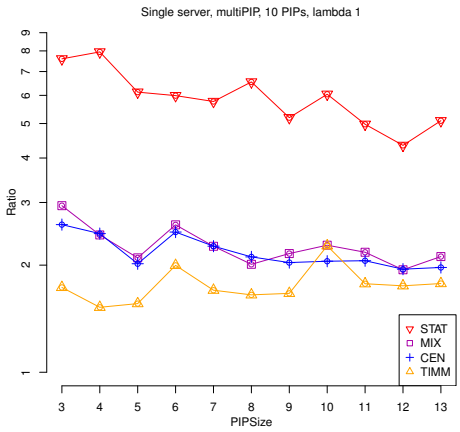


Figure 7.4: **Competitive ratio as a function of PIPSize, averaged over all** $p \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$

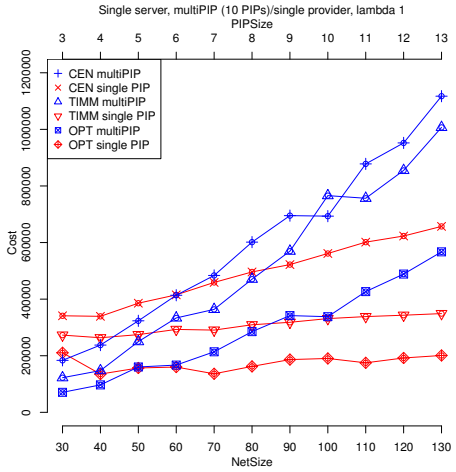


Figure 7.5: **Cost comparison (multi-PIP/singlePIP) in function of PIPSize/NetSize, averaged over all** $p \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$

TIMM.

Overall, TIMM performs very well for moderate p and even on average over all p for low λ. However, CEN/MIX improve for higher λ and adapt very well to different correlation values p. More specifically, they do not migrate in situations where a STAT shows good performance. They may therefore be preferable in scenarios with moderate to high λ and low to moderate request correlation, adapting to occasional spikes in p. Effectively, being adaptive they may be considered a suitable initial choice for new services with yet insufficiently known usage patterns.

Finally, a note regarding experiments on Rocketfuel topologies. Also there, the competitive ratios were low. For instance, on the autonomous system AS-7018 of `ATT` which consists of 115 nodes, and averaging over 50 runs at a runtime of 100, the optimal offline cost is 477.905648298, MIX has cost of 1179.85 and CEN has cost 825.81, which implies a competitive ratio is around 2.4 and 1.7, respectively.

7.5.5 Inter Provider Migration

As in the single provider case, Figure 7.4 shows the competitive ratio as a function of PIPSize averaged over all values of p. The first interesting observation is a counter intuitive improvement of the ratio for STAT. Figure 7.5 shows a comparison of cost developments for single PIP and multi-PIP experiments. It shows a drastic increase in costs for OPT explaining this phenomenon. The cost increase relates to

the different scenario setup. While zones were previously interconnected by a mesh structure, they are now interconnected as a ring. It turns out that while migration is more expensive in the multi-provider scenario, the increase in access cost is even more important to the extent that it often seems to be the dominating cost. OPT is still able to compensate by migration to a certain extent. But the average access cost from other zones and providers remain considerably increased for lower p. The same effect is visible for TIMM in both figures. Contrary to TIMM, CEN, MIX and OPT are more flexible in compensating by migration, and therefore appear to scale better with large PIPSizes. However, as OPT is able to identify more sensible migration goals, the increased migration costs can be assumed to affect its cost more than CEN/MIX.

These effects can also be seen in the detailed results shown in Figure 7.6. The upper pair of plots show the competitive ratio of CEN and TIMM as a function of p and PIPSize. It shows a behaviour similiar to the single provider experiments, though reduced in extent for lower p. Again, CEN emulates STAT when beneficial. With low p, remote requesters (often costing several 100 units due to the ring structure) are hardly avoidable. This drastic increase in access cost dominates the cost increase in inter-zone migrations. As an artifact of the round-robin attribution of zones to PIPs, migration only sometimes crosses more than one PIP border. This becomes clearly visible in the middle figures, where results are plotted in function of λ for a moderate $p = 0.6$. In a scenario where migration costs have only a limited effect on incurred access cost, the λ effect on TIMM is reduced, and λ loses its effect on migration amortization for CEN/MIX.

The lower plots of Figure 7.3 show the effect of inter-PIP migration cost factor x and the number of PIPs. As overall cost increases with x as expected, access costs seem slowly balanced and PIPSize (or migration goals) seem to matter again even at $p = 0.6$. Moreover, CEN seems to scale better with the number of PIPs. As every PIP is associated to one zone, and as TIMM chooses migration goals based on zones (not PIPs) only, its chances to choose the 'wrong' PIP increase. Also, TIMM will never migrate to a non-current zone to place itself between heavy hitting non-active zones. To these respects, the increased amount of PIPs corresponds to a reduced predictability of 'hot' subnets. It therefore can be considered to yield a basic intuition on TIMM's performance, if the order of *current* zones is less predictable.

Again, TIMM performs particularly well in low λ and moderate p cases. While CEN/MIX deliver stable results for a moderate $p = 0.6$ and low x, it is outperformed by TIMM in scenarios where migrations are of limited effect on access costs. This however changes with increasing inter-PIP migration costs, where it appears to scale better than TIMM.

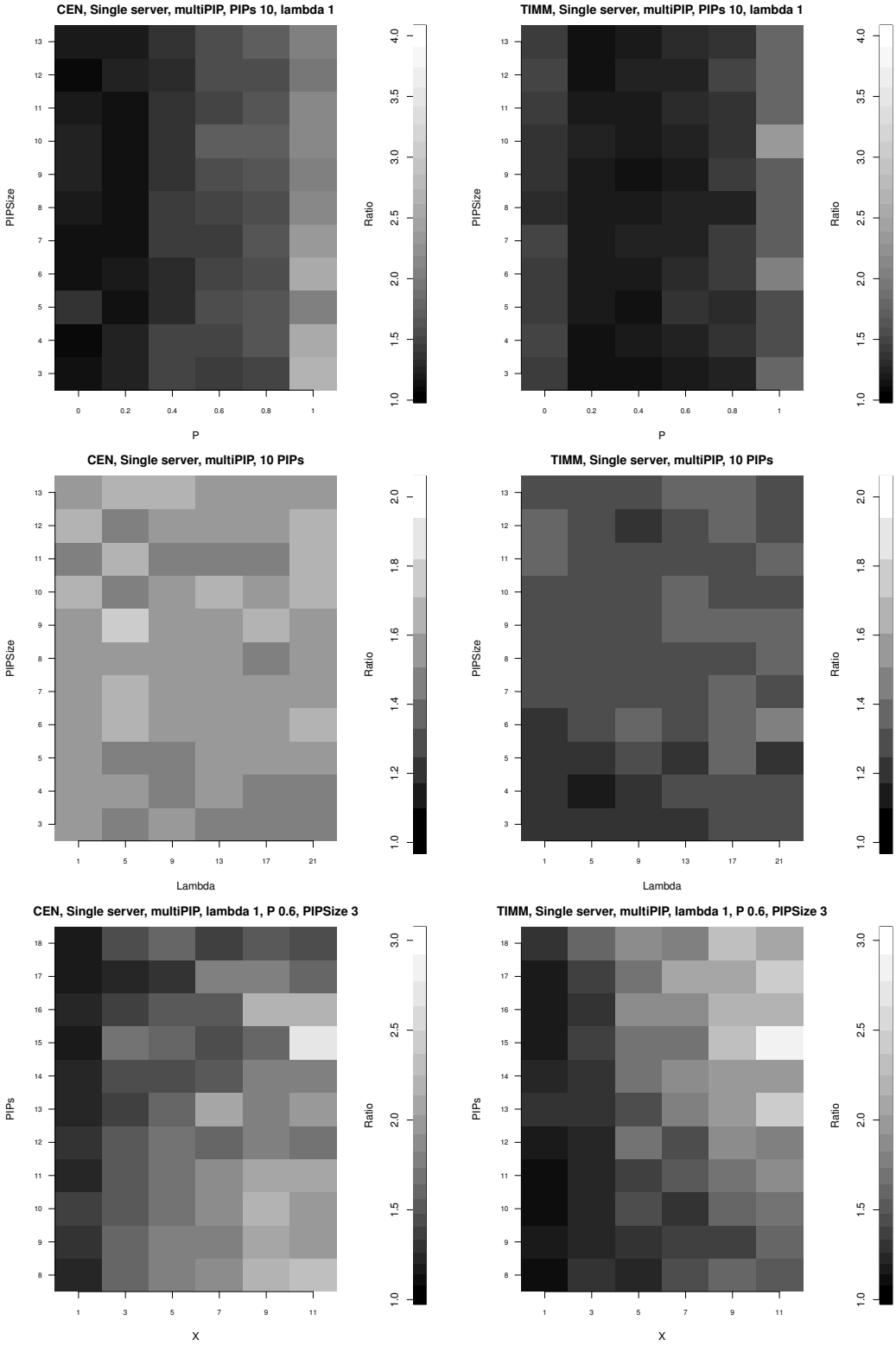


Figure 7.6: **Multi-provider results for different parameter sets**

7.5.6 Related Work

In this chapter, we tackle the question of how to dynamically embed or migrate virtual servers [99] in order to efficiently satisfy connection requests arriving online at any of the network entry points, and thus use virtualization technology to improve the quality of service for mobile nodes. The relevance of this subproblem of the general embedding problem is underlined by Hao et al. [73] who show that under certain circumstances, migration of a Samba front-end server closer to the clients can be beneficial even for bulk-data applications.

Migration and Online Algorithms. The possibility to migrate services without service interruption and even live has interesting implications and poses new challenges. For instance, see [41] for a recent attempt to model tradeoffs (in the cloud). To the best of our knowledge, [35] and [28] (for online server migration), and [57] (for online virtual network embeddings) are the only works to study network virtualization from an online algorithm perspective. The formal competitive migration problem is related to several classic optimization problems such as facility location, k-server problems, or online page migration. All these problems are a special case of the general *metrical task system* (e.g., [38, 39]) for which there is, e.g., an asymptotically optimal deterministic $\Theta(n)$-competitive algorithm, where n is the state (or "configuration") space; or a randomized $O(\log^2 n \cdot \log\log n)$-competitive algorithm given that the state space fulfills the triangle inequality: this algorithm uses a (well separated) tree approximation for the general metric space (in a preprocessing step) and subsequently solves the problem on this distorted space; unfortunately, both algorithmic parts are rather complex.

In the field of *facility location*, researchers aim at computing optimal facility locations that minimize building costs and access costs (see, e.g., [67] for an online algorithm). In [84], Laoutaris et al. propose a heuristic algorithm for a variant of a facility location problem which allows for facility migration; this algorithm uses neighborhood-limited topology and demand information to compute optimal facility locations in a distributed manner. In contrast to our work, the setting is different and migration cost is measured in terms of hop count. There is no performance guarantee. Zhang et al. [139] extend the problem formulation of [84] to server loads and prove a constant (static) approximation guarantee. However, the performance over time in a dynamic setting is unclear, and the benefits of dynamic resource allocation cannot be assessed. Moreover, in our more granular model, there are not only facility creation but also running costs and the notion of mobility of requests (i.e., a dynamic demand over time) is made explicit. In the field of *k-server problems* (e.g., [38]), an online algorithm must control the movement of a set of k servers, represented as points in a metric space, and handle requests that are also in the form of points in the space. As each request arrives, the algorithm must determine which server to move to the requested point. The goal of the algorithm is to reduce the total distance that all servers traverse. In contrast, in our model it is possible

to access the server remotely, that is, there is no need for the server to move to the request's position. The *page migration problem* (e.g., [34]) occurs in managing a globally addressed shared memory in a multiprocessor system. Each physical page of memory is located at a given processor, and memory references to that page by other processors are charged a cost equal to the network distance. At times, the page may migrate between processors, at a cost equal to the distance times a page size factor. The problem is to schedule movements on-line so as to minimize the total cost of memory references. In contrast to these page migration models, we differentiate between access costs that are determined by latency and migration costs that are determined by network bandwidth.

There is an intriguing relationship between server migration and *online function tracking* [36, 136]. In online function tracking, an entity Alice needs to keep an entity Bob (approximately) informed about a dynamically changing function, without sending too many updates. The online function tracking problem can be transformed into a chain network where the function values are represented by the nodes on the chain, and a sequence of value changes corresponds to a request pattern on the chain. In particular, it follows from [36] that already for some very simple linear substrate networks of size $n = \Theta(\beta)$, where β is the migration cost, no deterministic or randomized online algorithm can achieve a competitive ratio smaller than $\Omega(\log n / \log \log n)$.

Multi-Provider Aspects. Provisioning virtual network services across multiple providers is an interesting topic which is hardly explored in literature so far; notable exceptions are *PolyVINE* [45], a distributed coordination protocol to perform cross-provider embeddings, or *V-Mart* [138] that describes an auction framework for task partitioning. In the context of clouds, there exists interesting literature as well, e.g., *Reservoir* [105] explores the notion of a federated cloud in which providers lease excess capacity to others in need of temporary additional resources, similarly to electrical power grids.

7.6 Summary

In this chapter, we presented two online algorithms Cen and Mix for migration decisions of a single server, and sketched an extension to a multi-server variant. The algorithms aim to strike a balance between access and migration costs to achieve a provable logarithmic competitiveness.

We evaluated the algorithms empirically using artificial Erdös-Rényi graphs in a time zone scenario with a single and multiple providers. To this end, we formulated and implemented an optimal offline algorithm Opt. We compared their performance with a static placement STAT at the weighted center of the substrate network and a timed migration algorithm TIMM.

TIMM achieves formidable results in scenarios in which the request gravity center follows the expected pattern and the correlation of requests is sufficiently large. STAT on the other hand performs well for low correlation values, but shows very bad results with moderate and high correlation. We find that CEN and MIX however behave like STAT whenever it is beneficial, and show only a moderate increase in their ratio over the different parameter ranges. This stability in cost effectiveness may make them useful approaches in scenarios with unknown request patterns.

Additional experiments need to be done to verify CEN/MIX's adaptability benefit in other scenarios. But they seem competitive at low ratios and to provide a solution with limited volatility in results when faced with the choice between static migration patterns and static placements.

One possible extension we see for CEN and MIX is to make them aware of expected request patterns. This awareness can be used to weight registered access costs and thresholds to avoid longer delays in their following the time zone switch. While this reduces the provable competitiveness, it is likely to improve efficiency in many real-world scenarios.

Chapter 8

Prototype

In the context of this thesis, we implemented both a VNet prototype and modeled several scenarios with it. They serve two purposes: a feasibility study to identify real world issues, as well as a proof-of-concept implementation.

In the following, we first outline the scenarios we model with our prototype. Next, we describe its design with respect to components, interfaces, and database structure. After introducing the testbed environment the prototype is running in, we describe the realization of the scenarios we address and conclude with a short discussion of problems we encountered and possible prototype extensions.[1]

In the following, we first introduce the testbed environment. We then elaborate on the prototype's implementation and setup details. After describing the demonstrator, we provide insight into the lessons learned during the implementation process. We then conclude with a summary, containing a short discussion on problems encountered and possible prototype extensions.

8.1 Scenarios

To identify problems and to test feasibility of our concepts, we modeled the prototype and its environment to reflect the following scenarios:

Distributed management: We modeled two connected ISPs. One of the ISPs is substructured in a VNP and two PIP roles, whereas the other one takes only one PIP role. This allows us to test approaches and mechanisms for compatibility with substructuring, when VNP roles need to interact with both PIP and other VNP roles. To gain insight into possible issues with the configuration of virtual entities inside a virtualized environment, we virtualized several substrate hosts.

Video streaming use case: We realized the scenario of a streaming server migrated when service level usage patterns change. A streaming server always migrates to the location of the highest number of users. The VNet and substrate domain have little to no information about each other, and thereby are separated by

[1] For implementation details, such as configuration parameters and API specification, see [111].

a strict abstraction barrier. This scenario allows us insight into the problem of interaction between these two domains in situations when embedding needs to be adapted in response to service level events. Moreover, it shows the feasibility of our concepts in Chapter 7.

8.2 Design

This subsection first describes the prototype component design, provisioning interface, and database structure — in terms of functionality, subcomponents and used tools. For details, see the document's annex.

8.2.1 Components

Figure 8.1 shows the component design of our prototype. Substrate components correspond to the respective network elements, whereas all other components are part of the respective role's management node. While they can in principle be distributed, the interfaces from the provisioning to the mapping and from the mapping to the embedding component have been implemented as local function calls for the sake of simplicity. Every role (VNO (optional), VNP, PIP) has a *MySQL*[7] **Database** of the same base structure for storing state information on the current configuration of its network.

The **Provisioning Interface** consists of a *lighttpd*[14] web server exposing provisioning functionality to customers via *XMLRPC*[9]. The interface definition and backend scripts are split into a VNet negotiation part (*negotiate*.rb*) and a control part for virtual network elements (**provision*.rb*)[2].

The **Mapping Component** consists of a set of Ruby [17] scripts and a mathematical program solver (*CPLEX*). More specifically, the mapping scripts are the ones named {*pip,vnp*}*_map.rb*. The component maps new or modified VNets by

1. solving the logic clause defined by specified ConstraintGroups (see Chapter 4)[3],
2. extending the partial request to a full VNet specification as required and fixing Feature values corresponding to the ConstraintGroup solution in an OL1 graph (see Chapter 4),
3. formulating a MIP (see Chapter 5),
4. formulating a configuration (ML0 graph) based on the solver's output,

[2] Unless specified otherwise scripts are written in Ruby and reside in */scripts/vnonode*, */scripts/vnpnode*, and */scripts/snode*. Node configuration is stored in */etc/define.rb*, and functionality used by several roles is found in */scripts/common*.

[3] multiple solutions are tested iteratively, if they prove not embeddable

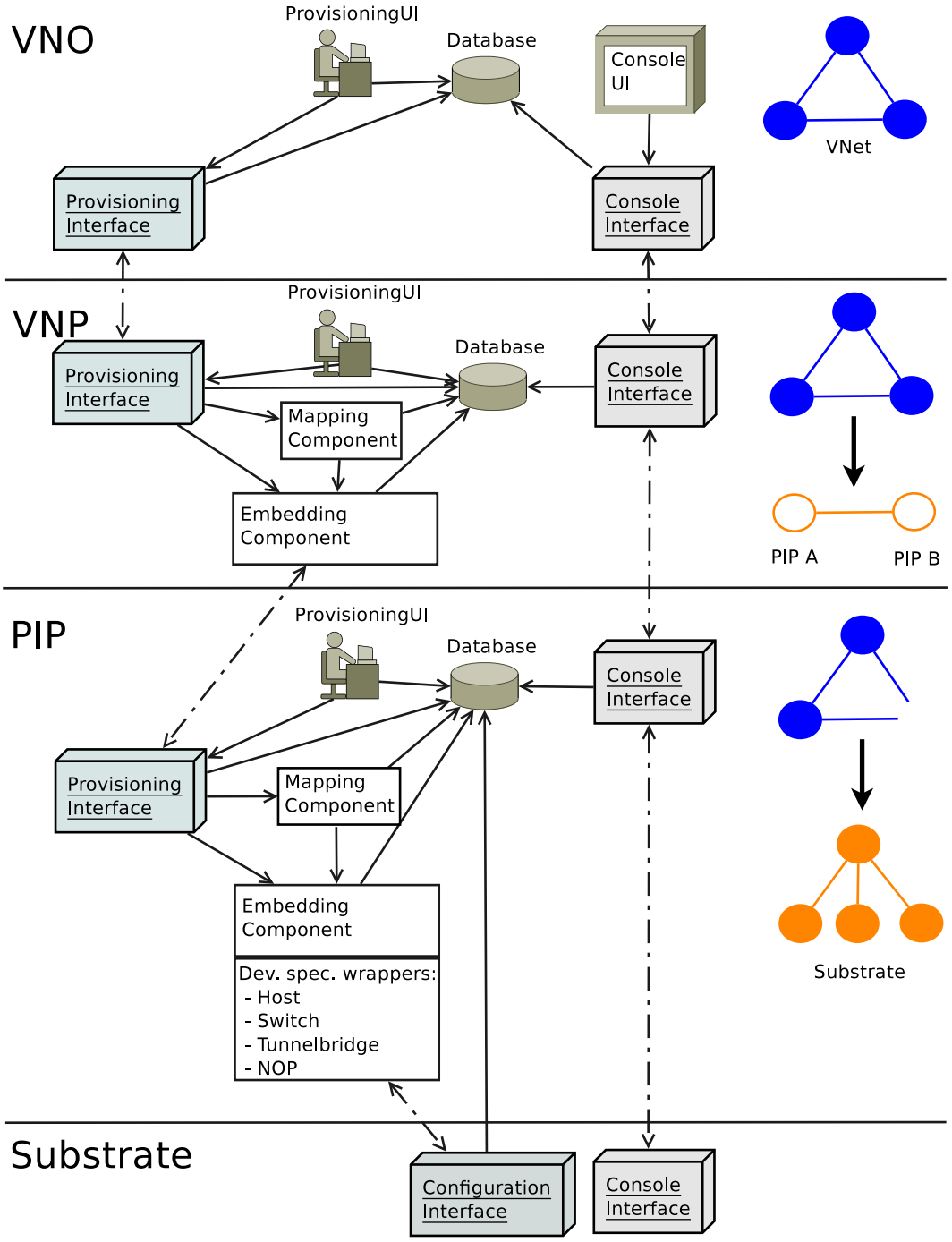


Figure 8.1: **Prototype components**

5. (VNP only) splitting up ML0 into ML1 subgraphs which correspond to the requests to the respective providers,

6. calling the **Embedding Component** to embed the configuration.

The **Embedding Component** consists of a set of Ruby scripts ({*pip,vnp*}*_embed.rb*) and a set of plugins supporting the Configuration Interfaces of different substrate device types. The current set of plugins interact with the substrate switch via *SNMP* [97], the hosts via *XMLRPC*. In order to test the mapping process, we provide a simulation mode. For the simulation mode, the embedding functionality is replaced by NoOps using the corresponding plugin. The latter is also used by PIPs to handle components mapped to remote providers. Such components may need to be considered to allow mapping of outgoing virtual links.

The prototype supports both a text and a graphical **Console Interface** for virtual hosts. The text interface is implemented via **_console.sh* scripts which are started by a *telnetd* agent. This script uses an agent (currently *telnet*) to connect to the appropriate Console Interface. A graphical interface can be requested via the Provisioning Interface. This request establishes a chain of *netcat* [8] instances, to which a *VNC* [125] client can connect.

Substrate **Console Interfaces** and **Configuration Interfaces** are device specific. The **Console UI** consists of either a *telnet* or a *VNC* client. The **Provisioning UI** is formed by a set of scripts. A GUI is under development.

8.2.2 Provisioning Interface

The provisioning interface is divided into two parts. A negotiation part and VNE control part. The negotiation part offers

create_vnet to request the instantiation of a VNet. It accepts a YAML [10] serialized VNet graph and returns a boolean success value. The requested network may be a part of a VNet which spans multiple providers to be embedded either as a whole or not at all. The resulting embedding is therefore considered tentative and is deleted after a timeout, unless confirmed. For debugging purposes, the prototype implementation in addition returns the database ID of the created OL0 graph as well as STDOUT and STDERR output.

confirm_request to confirm a VNet previously requested via create_vnet. It accepts a net_identifier, removes the 'preliminary' status, and returns a success status as well as STDOUT and STDERR output of the processes.

cancel_request to cancel a request sent via create_vnet, effectively removing it from the provider's database. It accepts a net_identifier and returns a success status as well as STDOUT and STDERR output.

delete_vnet to delete an instantiated VNet. It accepts a *v_net_identifier* of the created OL0 graph and, like create_vnet, returns a status value along with STDOUT and STDERR output.

modify_vnet to request a modified VNet. It currently accepts a *v_net_identifier* of the old OL0 graph along with a graph describing the future VNet. Return values correspond to those of create_vnet.

get_metrics to request a (coarse) summary of the provider's available (advertised) resources. Our prototype currently returns information about the total amount of RAM and HDD capacity available.

The control part allows to

start/stop/suspend/resume/powercycle NEs by specifying a ne_identifier and a net_identifier (start_vnode, stop_vnode, suspend_vnode, resume_vnode, powercycle_vnode). Return values are a boolean success information as well as STDOUT, STDERR output for debugging purposes.

start/stop/suspend/resume/powercycle VNets by specification of a net_identifier (start_graph, stop_graph, suspend_graph, resume_graph, powercycle_graph). These perform the respective action for all network elements in the VNet. The return values are identical to the VNode versions.

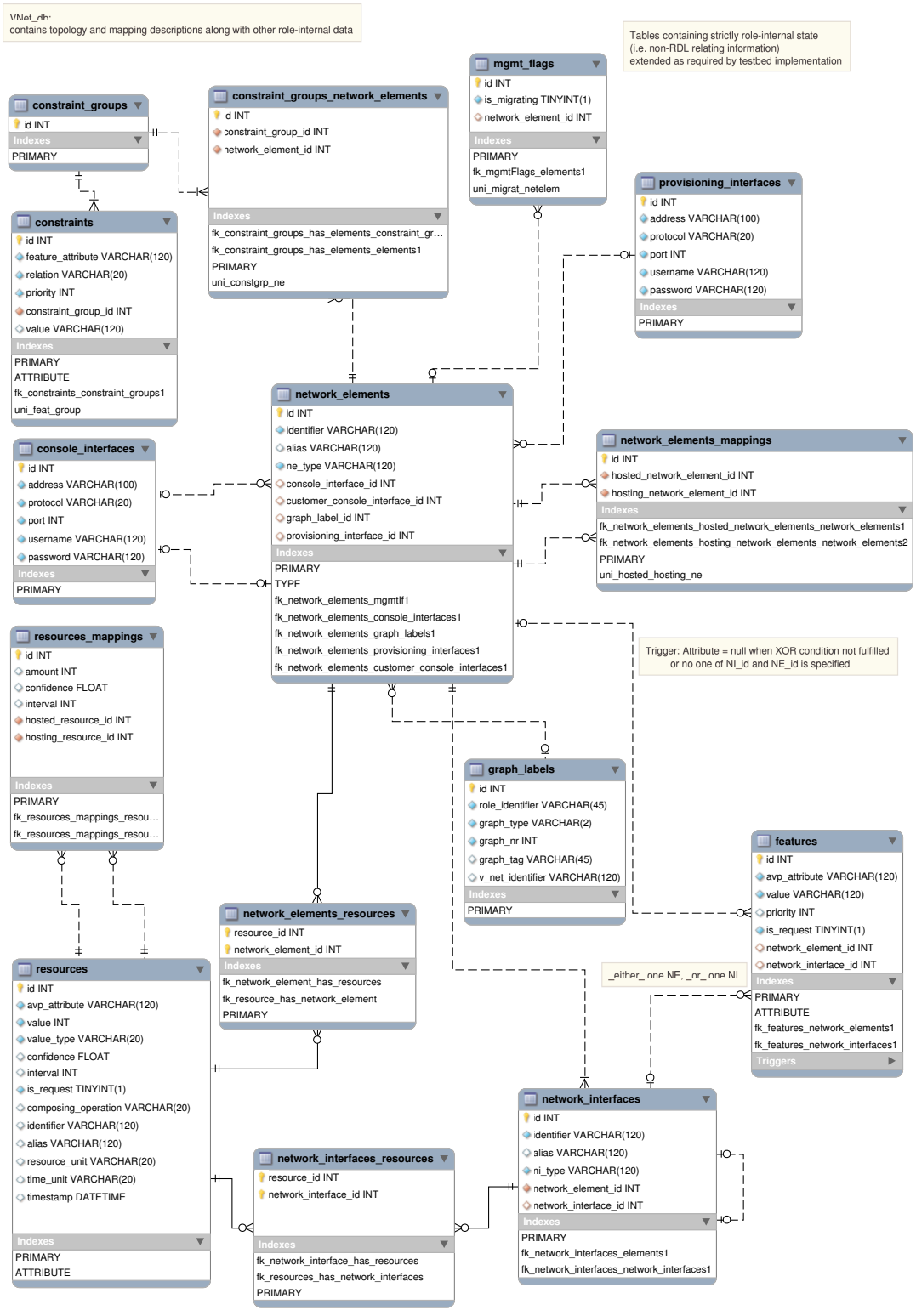
query NE and VNet status by specifying a net_identifier (graph_running) and an additional ne_identifier (node_running). Aside from STDOUT and STDERR output, these functions return *TRUE* when the specified network element (or all network elements of the given VNet) are active and running.

8.2.3 Database structure

The database structure is shown in Figure 8.2. It is closely related to the RDL discussed in Chapter 4. Every object described is reflected in a canonically named database table with the addition of a unique identifier field *id*. Naming conventions follow the requirements of ActiveRecord [13], which we use in our scripts to access the database. Hence, n:m relation tables are named after the tables they reference to, and the NetworkInterface field connectedInterface (see Chapter 4) is named *network_element_id*.

The additional table *mgmt_flags* allows for extensions with respect to state information required for provider operation. For example, the *is_migrating* field specifies whether the NE is currently unavailable due to migration. This is used by the ConsoleInterface to avoid unnecessary connection attempts.

Some fields carry slightly diverging names due to conflicts with reserved words inside ActiveRecord (*avp_attribute* instead of attribute) or for historic reasons (e.g., constraint solving implemented only for Features, and *constraint* field attribute called


VNet_db:
contains topology and mapping descriptions along with other role-internal data
Tables containing strictly role-internal state
(i.e. non-RDL relating information)
extended as required by testbed implementation
constraint_groups
id INT
Indexes
PRIMARY
constraint_groups_network_elements
id INT
constraint_group_id INT
network_element_id INT
Indexes
fk_constraint_groups_has_elements_constraint_gr...
fk_constraint_groups_has_elements_elements1
PRIMARY
uni_constgrp_ne
mgmt_flags
id INT
is_migrating TINYINT(1)
network_element_id INT
Indexes
PRIMARY
fk_mgmtFlags_elements1
uni_migrat_netelem
provisioning_interfaces
id INT
address VARCHAR(100)
protocol VARCHAR(20)
port INT
username VARCHAR(120)
password VARCHAR(120)
Indexes
PRIMARY
constraints
id INT
feature_attribute VARCHAR(120)
relation VARCHAR(20)
priority INT
constraint_group_id INT
value VARCHAR(120)
Indexes
PRIMARY
ATTRIBUTE
fk_constraints_constraint_groups1
uni_feat_group
network_elements
id INT
identifier VARCHAR(120)
alias VARCHAR(120)
ne_type VARCHAR(120)
console_interface_id INT
customer_console_interface_id INT
graph_label_id INT
provisioning_interface_id INT
Indexes
PRIMARY
TYPE
fk_network_elements_mgmtIf1
fk_network_elements_console_interfaces1
fk_network_elements_graph_labels1
fk_network_elements_provisioning_interfaces1
fk_network_elements_customer_console_interfaces1
network_elements_mappings
id INT
hosted_network_element_id INT
hosting_network_element_id INT
Indexes
fk_network_elements_hosted_network_elements_network_elements1
fk_network_elements_hosting_network_elements_network_elements2
PRIMARY
uni_hosted_hosting_ne
console_interfaces
id INT
address VARCHAR(100)
protocol VARCHAR(20)
port INT
username VARCHAR(120)
password VARCHAR(120)
Indexes
PRIMARY
resources_mappings
id INT
amount INT
confidence FLOAT
interval INT
hosted_resource_id INT
hosting_resource_id INT
Indexes
PRIMARY
fk_resources_mappings_resou...
fk_resources_mappings_resou...
Trigger: Attribute = null when XOR condition not fulfilled
or no one of NI_id and NE_id is specified
graph_labels
id INT
role_identifier VARCHAR(45)
graph_type VARCHAR(2)
graph_nr INT
graph_tag VARCHAR(45)
v_net_identifier VARCHAR(120)
Indexes
PRIMARY
features
id INT
avp_attribute VARCHAR(120)
value VARCHAR(120)
priority INT
is_request TINYINT(1)
network_element_id INT
network_interface_id INT
Indexes
PRIMARY
ATTRIBUTE
fk_features_network_elements1
fk_features_network_interfaces1
Triggers
network_elements_resources
resource_id INT
network_element_id INT
Indexes
fk_network_element_has_resources
fk_resource_has_network_element
PRIMARY
resources
id INT
avp_attribute VARCHAR(120)
value INT
value_type VARCHAR(20)
confidence FLOAT
interval INT
is_request TINYINT(1)
composing_operation VARCHAR(20)
identifier VARCHAR(120)
alias VARCHAR(120)
resource_unit VARCHAR(20)
time_unit VARCHAR(20)
timestamp DATETIME
Indexes
PRIMARY
ATTRIBUTE
either one NF, or one NI
network_interfaces
id INT
identifier VARCHAR(120)
alias VARCHAR(120)
ni_type VARCHAR(120)
network_element_id INT
network_interface_id INT
Indexes
PRIMARY
fk_network_interfaces_elements1
fk_network_interfaces_network_interfaces1
network_interfaces_resources
resource_id INT
network_interface_id INT
Indexes
fk_network_interface_has_resources
fk_resources_has_network_interfaces
PRIMARY

feature_attribute). Likewise, the hostedNetworkInterfacesList and hostingNetwork-InterfacesList of NetworkInterface are not implemented yet as such. The prototype relies instead on the *alias* field to retain mapping information.

8.3 Environment

The implementation and experiments were conducted in the Routerlab CITATION environment. More specifically, the environment contains

- several Sun X4150 machines featuring 8 Intel Xeon L5420 cores running at 2.5GHz, 16 GB RAM with 4 or 8 GBit Ethernet interfaces.
- a Cisco 4503 switch featuring 2 Mini-GBIC GBit interfaces and 96 GBit Ethernet ports which interconnects the Sun machines.

The Sun machines are running either KVM-devel-88 (Linux kernel 2.6.26.8) or Xen-common-3.4.1 (Linux kernel 2.6.18.8-xen) and OpenVSwitch 1.1.0pre2 [16]. The kernel versions are chosen for their compatibility with OpenVSwitch. The operating system (*OS*) is Ubuntu hardy.

8.4 Distributed management scenario

In order to capture additional real world issues, we decided to model a scenario with multiple VNP roles and recursive host virtualization. To this end, we modeled 4 entities, as pictured in Figure 8.3: One VNO, one VNP, and two provider (ISPs), one of which is substructured into a VNP role and two PIP roles.

Loadgen121 and *Loadgen123*, while serving as KVM virtualized substrate hosts *sn113* and *sn212*, host virtualized Xen substrate hosts. The management hosts of *PIP11*, *PIP12*, *PIP2* respectively export an NFS share *vdisks* to all managed substrate hosts in their domain. *vdisks* contains the disk images of the virtual hosts as well as the templates to create them. *nyc_sc1* is partitioned into logical switches by assigning a fixed set of ports and VLAN ids to each PIP. *Tunnelbridge* nodes provides layer 2 connectivity between the different PIPs.

We decided to include *TBR12* despite the fact that it is not actually interconnected to any other PIP by OpenVPN. The rationale for this step is that the initial Open-VSwitch versions we used on the substrate hosts did not support VLAN bridging. As the *Tunnelbridge* nodes ran the regular Linux bridge, they were the only hosts capable of hosting virtual switches.

A separate VLAN serves as management network, interconnecting provisioning, configuration, and console interfaces. *TBR1* is interconnected via one layer 3 OpenVNP connection (management plane) and one layer 2 OpenVPN connection (data plane)

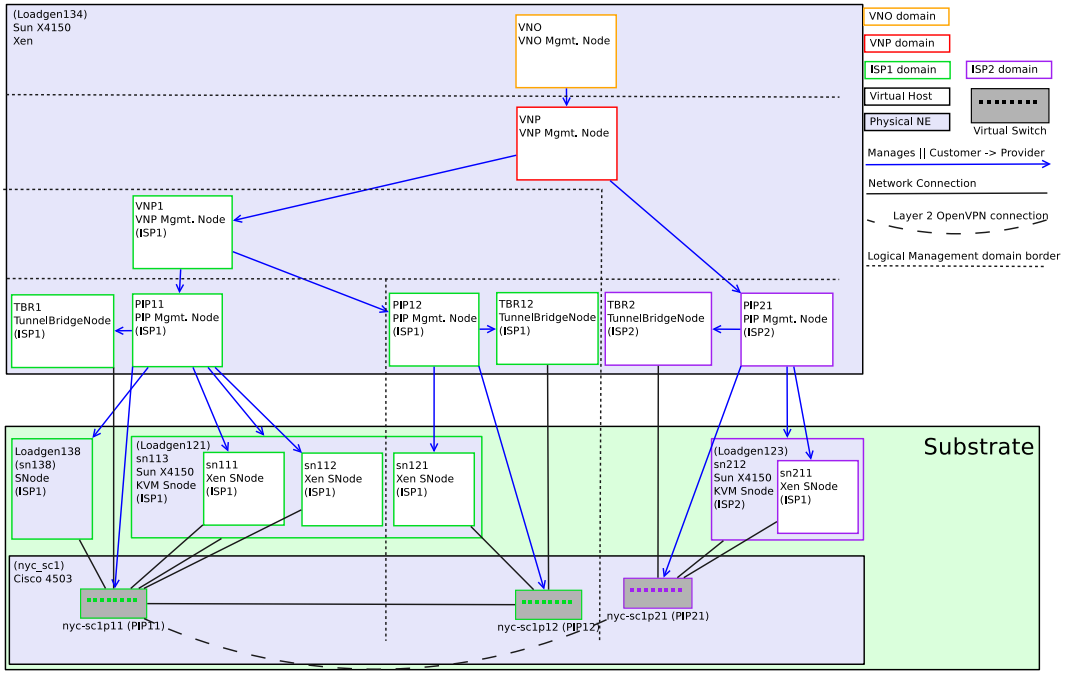


Figure 8.3: **Prototype setup (distributed management scenario)**

with a similiar environment running at Docomo Eurolabs Munich. *VNP* may request resources from their exposed VNP role, while the latter is allowed to request resources from *VNP1*.

8.5 Video streaming scenario

In this scenario, we consider a video streaming server and a set of clients in different locations (see Figure 8.4). Clients which are local with respect to the server's position experience lower latency of the live video stream. In an effort deliver the best service, the server migrates to the *center of gravity*, i.e., the location with the highest number of active clients. To this end, as mentioned in Section 8.1, the service (running inside a VNet) needs to interact with components managing the embedding (in the substrate).

Three machines in our PIP serve as access point location (defined by position statements). The accesspoints on these machines are connected to the VNet on their *eth0* interface, and to a respective access network via *eth1*. A webbased service interface allows the enabling or disabling of video streams to the respective customers, represented by 9 VLC [18] clients running on 3 laptops. Since at the time of writing, the prototype implementation does not yet support Access Network management,

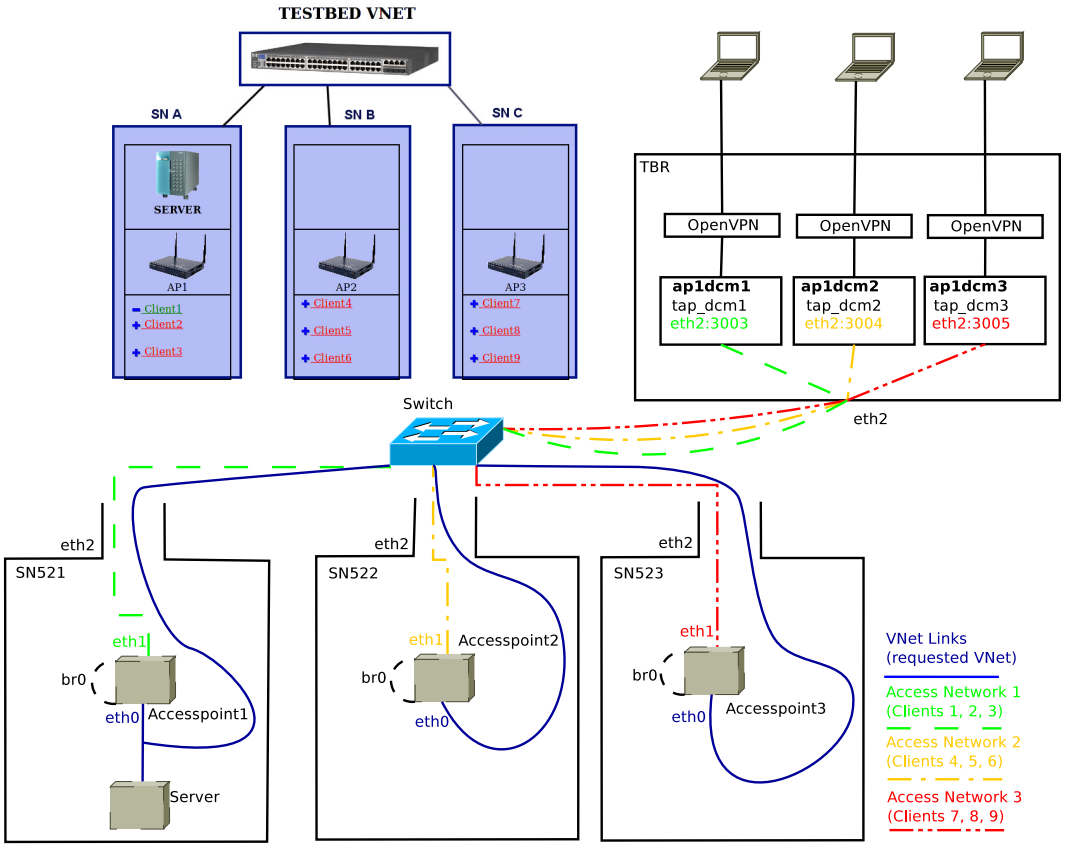


Figure 8.4: **Prototype setup (video streaming scenario)**

the three access networks, to which the clients connect via OpenVPN (Layer 2), are configured manually.

Logically, the web portal should reside inside the VNet. A process on the VNO's management node should observe its service state, e.g., via ConsoleInterface connection. This process should then request VNet modifications via the provider's ProvisioningInterface. For access convenience however we decided to implement the mentioned portal and logic on the management node and used the ConsoleInterface to configure the service instead. Note that we merely shift the location in which the ConsoleInterface is used as communication channel. Neither did we provide either party with knowledge about the respective other domain, nor would a shift into the VNet be impossible. The service (i.e., the video streaming) is configured by modifying *iptable* [15] filter rules on the server, barring or allowing the video streams.

When a shift in the center of gravity is detected, the VNO's decision logic requests a VNet with modified properties via *modify_vnet*. In our demonstrator, the preformulated requests differ in constraints requiring co-location of the server with the

target access point. This enables the PIP to deduce a required migration and adjust the embedding as required by the service inside the VNet.

8.6 Lessons learned

As previously mentioned, implementing the prototype served as reality-check in several respects. On one hand, it provided a proof of concept, showing feasibility of our approaches. On the other hand, it confronted us with detailed problems, and thereby helped us to refine conceptual parts of this thesis. This section provides a short overview on the most important lessons learned on different levels.

8.6.1 Virtualization mechanisms

Our prototype implementation includes the use of recursive virtualization. As outlined above, we ran paravirtualizing Xen servers inside hardware virtualized KVM slices. While this works in principle, our experience shows that stability of such setups is still far from production grade status. Especially after prolonged periods of inactivity, the Xen instances tended to crash and needed to be rebooted via console of the physical machine.

Likewise, the interface bridge via OpenVSwitch often got into an inconsistent state upon reboot of the physical machines. In this case, packets were forwarded only in one direction, but packets into the opposite direction would not register on the interface of the physical server, until OpenVSwitch was restarted.

Moreover, we found that we had to disable the spanning tree protocols for VNet data plane vlans on the Cisco switch. The reason for this was that the switch would block respective ports when it registered incoming packets on one VLAN it sent on a different VLAN.

Many mechanisms thus may be sufficiently stable for use in some scenarios. However, we conclude that they still need to undergo more development and debugging to allow taking full advantage of possibilities offered by the concept of virtualization.

8.6.2 Protoype software

The prototype's functionality is focused on management - that is state-keeping and the configuration of existing components. In our first steps to gain insight into the topic, we aimed to minimize the implementation overhead and started with a set of shell scripts, re-using available tools. However, as this approach would not scale well with functionality and had to compensate for many quirks in the diverse used toolset, we soon switched to the current programming language.

Again, our choice for Ruby was motivated by an effort to limit implementation overhead. With the "ActiveRecord" extension and YAML/XMLRPC support, it provided excellent means of simplifying database access and communication between management nodes.

Unfortunately, multiple practical problems would not make us suggest this language in a reimplementation effort. Most importantly, many implementations exhibit performance problems in terms of memory management and the different implementations support only different - partially incompatible - versions of YAML. Garbage collection proved a major problem in runtime when evaluating our MIP, as the formulation of the problem would consume large amounts of memory. Only jRuby proved sufficiently performant for this, while unfortunately being less stable.

8.6.3 VNet specification

Maybe most importantly, the prototype showed us the importance of many aspects in modeling resources and representing state. As an example, consider that networked file systems, such as NFS, facilitate live migrations, and therefore are likely to exist in many substrate environments. Splitting up such resources, e.g., assigning slices to substrate hosts, may make feasible solutions appear invalid in the mapping process. Likewise, using NFS resources or tunnels in inter-provider setups highlighted the importance of supporting mapping on both NetworkElement/NetworkInterface and Resource level.

One lesson learned on this level however is connected to an inconvenience in mapping VNet resources on aggregate elements in the substrate's representation due to our chosen Resource/Feature attribute hierarchy. In our prototype, we chose to prefix resource attributes with the type of the entity (NE/NI) their are associated with. When mapping VNetsonto a topology of PIPs on VNP level, this led to a higher complexity in identifying suitable (i.e., compatible) mappings. A similar problem may occur when connecting shared resources to entities of different types.

We therefore do not recommend to chose this approach to structuring the attribute hierarchy. Instead, we would put emphasis on standardization, reserving a private prefix for ad-hoc extensions.

8.7 Summary

In this chapter, we described the proof-of-concept prototype. The prototype was realized in our Routerlab testbed environment and is interconnected with a similiar installation at our cooperation partner Docomo Eurolabs Munich. It can represent multiple providers in a subcontracting scenario with a partially virtualized substrate.

Some of the limitations that we had to cope with are that the virtualized Xen substrate hosts were not perfectly stable and performance was decreased compared to the xen instances running on loadgen138. Nonetheless, it highlights that reselling of virtualized resources in a virtualized substrate may become viable. While our prototype does not yet support wide area live migration, this is in principle possible. The current prototype does only enforce limits on RAM and disk resource usage, but not on virtual links. We imagine that OpenFlow replaces VLAN functionality in the near future.

In the video streaming scenario, we consider a live video streaming service inside a VNet with explicit control of server migration. The server position is adapted according to the current service situation. This adaptation can be requested even in presence of strict constraints regarding information exchange between VNet provider and customer. In our scenario, communication between the in- and the outside of the VNet is revised via the ConsoleInterface. It may however also be facilitated via a gateway node connected to multiple network elements.

Chapter 9

Conclusion

The current internet suffers from ossification in the underlying infrastructure. Virtual networks may not only be a promising approach to introduce new technologies and allow for co-existance of mutually exclusive network design goals. It also has potential to lower the financial barriers for field-tests or introduction of novel services, e.g., for startup companies. Moreover, virtualization already nowadays serves as an enabler for more dynamic resource management, and thereby has direct impact on CAPEX and OPEX benefits. This thesis addressed the topic of virtual network management.

We propose a management framework focussing on real-world constraints in the context of multi-provider scenarios. Our framework aims at allowing for technical flexibility while abstracting forseeable economic tussles from the technical details. In fact, mangement itself may turn out to be a viable business field. We identify granular management roles motivated by tasks found in real world network management. These roles can be spliced in optimization efforts within an enterprise or an organization. Still they allow to split all essential tasks amongst different entities in a structured manner. As it is the necessary basis of the role's customer-provider relationships, our interfaces strictly rely on requirement driven information exchange. This again does not stop organizations to optimize management by extending the interfaces by multilateral agreements. We argue that limiting assumptions on information exchange yields support for a wide variety of information exchange policies without breaking the framework itself. The nature of the VNP role enables both internal substructuring, e.g., inside a large company, and subcontracting scenarios. We outline an Out-of-Vnet access to support abstraction on every level, and sketch possible approaches for end user terminal attachement.

With respect to inter-provider communication, we design a resource description language, which allows for both concise descritions and vagueness. This facilitates both inter-provider communication about requirements and communication inside a substrate provider about current state. Customers on the one hand may or may not be interested in arbitrary details. Moreover, overspecification has a direct impact on the provider's management flexibility and consequently on its benefits. Providers on the other hand may offer services on arbitrary abstraction levels. They may wish to describe products in marketing terms only loosely or not at all following technical conventions. These requirements on flexibility and extensibility form the basis

for design choices of our RDL. Allowing to omit arbitrary descriptions or even network components, and the support for white- or blacklisting adresses these issues. To enable different levels of information exchange, we allow to formulate recursive mappings to enable closer cooperation between roles. Moreover, aside from properties of individual network elements and topological details, we support non-topological constraints, such as co-location or binary compatibility. The underlying data structure aims to facilitate both graph algorithm application in the presence of multi-homed links, and translation into full-scale ontologies for automated reasoning.

With respect to the assignment of virtual network elements to substrate network elements, we formalize the mapping problem as MIP. Our focus lies on the integration of real-world aspects into the formulation and our solution is designed for an offline optimization of long lasting networks. The MIP is based on the multi-commodity flow problem of bandwidth resource allocation between virtual nodes. Following the RDL's description model, it can be applied not only on the PIP level, but also on the VNP, or VNO level. By the same means, we support multi-homed shared communication channels and allow to cross-map resource types (e.g., bandwidth to CPU resources on switches or routers). We describe and account for migration costs, allowing reconfiguration of previously placed virtual resources. Moreover, our MIP supports co-hosting of partial VNet-topologies on single substrate hosts, and we outline several potential extensions, e.g., with respect to distributed node embedding. While scalability of the basic approach seems limited, it can be useful in specific scenarios. Assuming a series of steps in mapping virtual networks on the level of various roles, we argue that it merely needs to scale to the scope of the respective role: On a PIP level, it can serve on small or medium router sites, whereas on a VNP level, it can serve to map VNets to sites or providers. Groups of nodes in data centers with largely homogeneous components can be aggregated to compound network elements in an approximation effort.

Bearing this in mind, we ask the question on the benefit of different approximation approaches on mapping performance. Considering a scenario with strict resource guarantees and no substrate overbooking, we measure mapping quality both by the number of resources required for embedding and the number of supported virtual nodes. We explore both solver related approximations, such as using a different GAP parameter, and problem formulation approximations, such as clustering. We find that even when the solvers are not used to calculate optimal solutions, they often produce good results. Indeed, they provide quality bound guarantees on intermediate solutions, when interrupted. In our scenario, the best results were achieved by modifying the GAP parameter, monitoring the development of GAP, and clustering.

One of the advantages of VNets is that they can shrink and expand dynamically. Our architecture and mapping support such dynamics. This is sufficient when the effects of resource allocations are predictable. However, they not always are. In this case, online algorithms can help to make decisions on where and when to migrate.

As a first step, we consider a single server scenario with defined access and migration costs. For this, we propose two algorithms controlling migration. These algorithms guarantee logarithmic competitiveness for both single-PIP and multi-PIP scenarios. In addition, we sketch an extension to a multiserver version. Depending on the role context, input and output may be based on different abstractions, and migration control may be implicit. Nonetheless, they can be applied to all abovementioned roles. We compare the presented algorithms against an optimal offline algorithm as well as two other approaches: a static placement, as well as an approach for scheduled migrations in a time zone scenario. Our initial experiments show a stable performance at a low to moderate ratio. The online algorithms migrate only when beneficial and adapt well to different parameter ranges. They are a promising choice for new services where usage patterns are either highly volatile or simply not yet known.

Following our aim for real-world applicability, we implemented the basic concepts and elements in a prototype and experimented with various scenarios. This serves the dual purpose of a reality check of our assumptions and sources of real world issues, which might otherwise be lost. The prototype shows the functioning of our VNet architecture, and uses our RDL and mapping approach in its operation. The scenarios show the feasibility of migration-triggering interaction between the service domain inside a VNet, and substrate operation. Our prototype is distributed across our routerlab environment and facilities located at Docomo Eurolabs in Munich.

9.1 Outlook

While we do not expect fully customizable multi-provider VNets to appear overnight, we see both technical and economic migration paths towards this technology. On the technical side, providers can introduce virtual networks either as slices in their current internet infrastructure, or encapsulate the legacy Internet in one VNet independent of others. On the economic side, the opportunities for CAPEX/OPEX benefit increase may make the introduction of VNets interesting for administration of owned infrastructure (single-PIP scenario). We expect large providers to substructure their infrastructure and thus implicitly use mechanisms and interfaces from multi-PIP scenarios. These can subsequently be used for cooperations with affiliate partners or daughter companies. At this point, the role of VNP may become an attractive business field and open the door for multi-provider VNets. Indeed, in 2012, Google announced that they intend to deploy SDNs on a large scale[89] and renting, e.g., Cloud resources from multiple providers is already reality [51]. Moreover, during the writing of this thesis, Amazon started to extend its Cloud services by allowing VPN access and the definition of subnets on a star topology by which hosts are interconnected [11]. Nonetheless, many issues and trade-offs remain to be investigated in the context of VNets, such as:

Information exchange: Many administrative details, such as topology and specific resource capacities, are considered business secrets. In principle, VNet negotiation mechanisms can be based on answering resource requests from customers with an answer which contains nothing more than a simple acknowledgement or refusal. This however is unlikely to be the most overall efficient solution. The apparent trade-off in negotiation efficiency and the protection of business secrets leads to the question on what amount and kind of information should be provided on the provider's behalf.

Price of specification: Likewise, as pointed out in Chapter 4, every request constraint reduces a provider's flexibility for optimization. This trade-off has to be investigated and characterized under both technical and economic aspects, and incentive-driven need to be devised.

Inter-VNet operations: Communication between VNets entail a number of issues: Not only protocol (e.g., address) incompatibilities need to be tackled, but isolation and security properties have to be guaranteed. Moreover, it is not clear whether terminals, which usually are under control of end users, can be stopped from acting as undesired gateways.

Fault detection and debugging: VNets require additional mechanisms which may be error prone. Finding the source of failures or a lack of performance in the presence of abstraction barriers introduced by virtualization is likely to be non-trivial. Strategies to identify and localize problems and their causes need to be devised.

Security: Unlike physical hardware, virtual infrastructure in multi-provider environments cannot be locked away in racks accessible only to their owners. They are — and need to be — remotely accessible on the most basic level. Bugs in virtualization mechanisms and malicious providers are likely to have serious impact on an even wider scale than in traditional networks. Especially measurement and debugging is not trivial to handle in the case of malicious or colluding entities.

Appendix A

FleRD XML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:group name="feature_list">
      <xs:sequence>
         <xs:element name="Feature" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="attribute" type="xs:string" minOccurs="1" maxOccurs="1"/>
                  <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
                  <xs:element name="priority" type="xs:integer" minOccurs="1" maxOccurs="1"/>
                  <xs:element name="request_flag" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:group>

   <xs:group name="network_interface_list">
      <xs:sequence>
         <xs:element name="NetworkInterface" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="id" type="xs:QName" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element name="identifier" type="xs:string" minOccurs="1" maxOccurs="1"/>
```

```
    <xs:element name="alias" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="ni_type" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="connectedInterface" type="xs:QName" minOccurs="0" maxOccurs="1"/>
    <xs:element name="hostedNetworkInterface" type="xs:QName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="hostingNetworkInterface" type="xs:QName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="resource" type="xs:QName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="features" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:group ref="feature_list"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:sequence>
</xs:group>

<xs:group name="topology_graph">
  <xs:sequence>
    <!--
        GraphLabel
    -->
    <xs:element name="GraphLabel" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="id" type="xs:QName" minOccurs="1" maxOccurs="1"/>
          <xs:element name="role_identifier" type="xs:string" minOccurs="1" maxOccurs="1"/>
          <xs:element name="graph_type">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="OL"/>
                <xs:enumeration value="ML"/>
                <xs:enumeration value="UL"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
```

```
      <xs:element name="graph_nr" type="xs:integer" minOccurs="1" maxOccurs="1"/>
      <xs:element name="tag" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="net_identifier" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <!-- isn't allowed to exist w/o NE referencing it - not required to reference to NEs (also: symmetric) -->
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!--
   NetworkElement
-->
<xs:element name="NetworkElement" minOccurs="1" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="id" type="xs:QName" minOccurs="1" maxOccurs="1"/>
      <xs:element name="identifier" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="alias" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="ne_type" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="graphLabel" type="xs:QName" minOccurs="0" maxOccurs="1"/>
      <xs:element name="hostedNetworkElement" type="xs:QName" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="hostingNetworkElement" type="xs:QName" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="resource" type="xs:QName" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="features" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:group ref="feature_list"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="networkInterfaces" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:group ref="network_interface_list"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="customerConsoleInterface" type="xs:QName" minOccurs="0" maxOccurs="1"/>
      <xs:element name="providerConsoleInterface" type="xs:QName" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<!--
    Resources
  -->
<xs:element name="Resource" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
  <xs:sequence>
    <xs:element name="id" type="xs:QName" minOccurs="1" maxOccurs="1"/>
    <xs:element name="identifier" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="alias" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="attribute" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="value" type="xs:float" minOccurs="1" maxOccurs="1"/>
    <xs:element name="resource_unit" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="value_type" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="confidence" type="xs:float" minOccurs="0" maxOccurs="1"/>
    <xs:element name="confidence_interval" type="xs:integer" minOccurs="0" maxOccurs="1"/>
    <xs:element name="time_unit" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="composing_operation" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="request_flag" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
    <xs:element name="timestamp" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<!--
   ConstraintGroups
  -->
<xs:element name="ConstraintGroup" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="networkElement" type="xs:QName" minOccurs="2" maxOccurs="unbounded"/>
     <xs:element name="Constraint" minOccurs="1" maxOccurs="unbounded">
       <xs:complexType>
         <xs:sequence>
           <xs:element name="attribute" type="xs:string" minOccurs="1" maxOccurs="1"/>
           <xs:element name="value" type="xs:string" minOccurs="0" maxOccurs="1"/>
           <xs:element name="relation" type="xs:string" minOccurs="1" maxOccurs="1"/>
           <xs:element name="priority" type="xs:integer" minOccurs="1" maxOccurs="1"/>
```

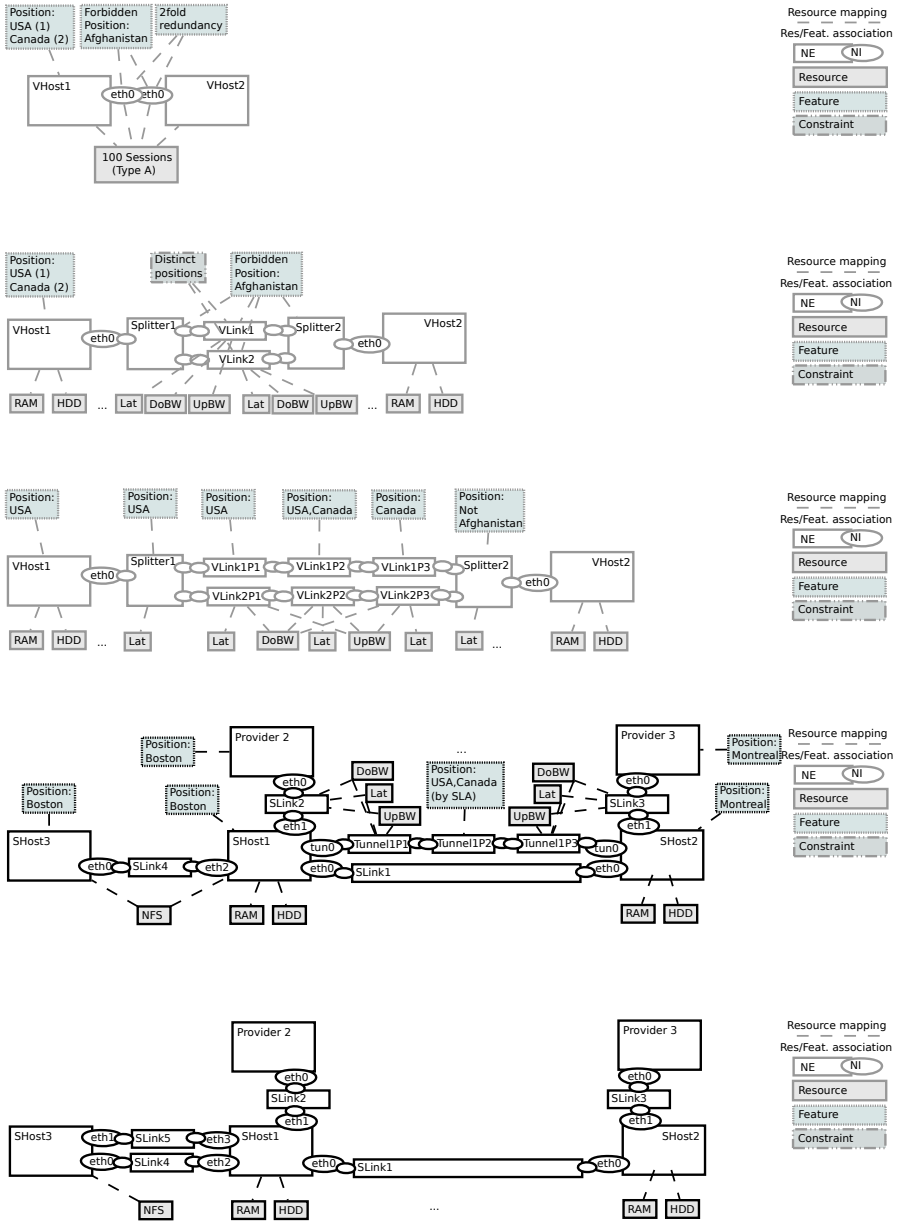
```
    </xs:sequence>
      </xs:complexType>
        </xs:element>
          </xs:sequence>
            </xs:complexType>
              </xs:element>
                </xs:sequence>
                  </xs:group>

                  <xs:element name="root">
                    <xs:complexType>
                      <xs:sequence>
                        <!--
                            ConsoleInterfaces
                        -->
                        <xs:element name="ConsoleInterface" minOccurs="0" maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="id" type="xs:QName" minOccurs="1" maxOccurs="1"/>
                              <xs:element name="address" type="xs:string" minOccurs="1" maxOccurs="1"/>
                              <xs:element name="transport_protocol" type="xs:string" minOccurs="1" maxOccurs="1"/>
                              <xs:element name="port" type="xs:integer" minOccurs="1" maxOccurs="1"/>
                              <xs:element name="username" type="xs:string" minOccurs="1" maxOccurs="1"/>
                              <xs:element name="password" type="xs:string" minOccurs="1" maxOccurs="1"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <!--
                            Topology graphs
                        -->
                        <xs:element name="Topology" minOccurs="1" maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:group ref="topology_graph"/>
                          </xs:complexType>
                        </xs:element>
                        <!--
```

```
    ResourceMappings
    -->
    <xs:element name="ResourceMapping" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="value" type="xs:float" minOccurs="0" maxOccurs="1"/>
          <xs:element name="resource_unit" type="xs:string" minOccurs="0" maxOccurs="1"/>
          <xs:element name="confidence" type="xs:float" minOccurs="0" maxOccurs="1"/>
          <xs:element name="confidence_interval" type="xs:integer" minOccurs="0" maxOccurs="1"/>
          <xs:element name="time_unit" type="xs:string" minOccurs="0" maxOccurs="1"/>
          <xs:element name="hostedResource" type="xs:QName" minOccurs="1" maxOccurs="1"/>
          <xs:element name="hostingResource" type="xs:QName" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Appendix B

FleRD example



```
<?xml version="1.0" encoding="UTF-8"?>

<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="flerd.xsd"
  xmlns:p="/flerd/example1">
  <!-- Console interfaces -->
  <ConsoleInterface>
    <!-- Console Interface offered to one specific customer
      - assumed to multiplex console requests
      - in our prototype: mgmt node -->
    <id>p:cci1</id>
    <address>192.168.1.1</address>
    <transport_protocol>tcp</transport_protocol>
    <port>2000</port>
    <username>customer1</username>
    <password>customerpass1</password>
  </ConsoleInterface>
  <ConsoleInterface>
    <!-- Console Interface for virtual elements hosted on SHost1 -->
    <id>p:pci1</id>
    <address>192.168.2.1</address>
    <transport_protocol>tcp</transport_protocol>
    <port>2000</port>
    <username>myusername1</username>
    <password>mypass1</password>
  </ConsoleInterface>
  <ConsoleInterface>
    <!-- Console Interface for virtual elements hosted on SHost2 -->
    <id>p:pci2</id>
    <address>192.168.2.2</address>
    <transport_protocol>tcp</transport_protocol>
    <port>2000</port>
    <username>myusername2</username>
    <password>mypass2</password>
  </ConsoleInterface>
  <ConsoleInterface>
    <!-- Console Interface for virtual elements hosted on SHost3 -->
    <id>p:pci3</id>
    <address>192.168.2.3</address>
    <transport_protocol>tcp</transport_protocol>
    <port>2000</port>
    <username>myusername2</username>
    <password>mypass2</password>
  </ConsoleInterface>



  <!-- actual topology graph descriptions -->
  <Topology>   <!-- OL0 - customer request -->
    <!-- not stating any embedding state annotations (optional)
      - relevant to the customer in this example graph
      - for readability's sake -->
    <!-- H-H -->
    <GraphLabel>
```

```
    <!-- this is a request on behalf of our customer1
      - but we are PIP1
      - and it is OL0 for us
      - while it may be, e.g., ML0 for our customer -->
    <id>p:c1n1o0gl1</id>
    <role_identifier>PIP1</role_identifier>
    <graph_type>OL</graph_type>
    <graph_nr>0</graph_nr>
    <net_identifier>VNet1</net_identifier>
  </GraphLabel>
  <NetworkElement>
    <id>p:c1n1o0h1</id>
    <identifier>VHost1</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:c1n1o0gl1</graphLabel>
    <hostingNetworkElement>p:c1n1o1h1</hostingNetworkElement>
    <!-- X: shared virtual resource -->
    <resource>p:c1n1o0r1</resource>
    <features>
      <!-- X: whitelist
        - this node may be hosted in the US (preferred) or Canada -->
      <Feature>
        <attribute>/position/continent/country</attribute>
        <value>/NA/USA</value>
        <priority>1</priority>
        <request_flag>true</request_flag>
      </Feature>
      <Feature>
        <attribute>/position/continent/country</attribute>
        <value>/NA/Canada</value>
        <priority>2</priority>
        <request_flag>true</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1o0h1ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <!-- X: omitted component:
          - not interested in link
          - p:c1n1o0h1ieth0 directly connected to VHost2:eth0 -->
        <connectedInterface>p:c1n1o0h2ieth0</connectedInterface>
        <hostingNetworkInterface>p:c1n1o1h1ieth0</hostingNetworkInterface>
        <resource>p:c1n1o0r1</resource>
        <features>
          <Feature>
            <!-- this connection should be failsafe (simple redundancy) -->
            <attribute>/link/generic/redundancy</attribute>
            <value>2</value>
            <priority>1</priority>
            <request_flag>true</request_flag>
          </Feature>
          <!-- X: blacklist:
```

```
            - no link part may be hosted Afghanistan -->
          <Feature>
            <attribute>/not/position/continent/country</attribute>
            <value>/Asia/Afghanistan</value>
            <priority>1</priority>
            <request_flag>true</request_flag>
          </Feature>
        </features>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:c1n1o0h2</id>
    <identifier>VHost2</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:c1n1o0gl1</graphLabel>
    <hostingNetworkElement>p:c1n1o1h2</hostingNetworkElement>
    <resource>p:c1n1o0r1</resource>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1o0h2ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o0h1ieth0</connectedInterface>
        <hostingNetworkInterface>p:c1n1o1h2ieth0</hostingNetworkInterface>
        <resource>p:c1n1o0r1</resource>
        <features>
          <Feature>
            <attribute>/link/generic/redundancy</attribute>
            <value>2</value>
            <priority>1</priority>
            <request_flag>true</request_flag>
          </Feature>
        </features>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <Resource>
    <!-- X: abstract resource
      - not interested in actual resources
      - requesting support for X sessions per second of type A -->
    <id>p:c1n1o0r1</id>
    <attribute>/service/A/concurrent_sessions</attribute>
    <value>100</value>
    <value_type>minimum</value_type>
    <confidence>0.95</confidence>
    <confidence_interval>1</confidence_interval>
    <time_unit>s</time_unit>
    <request_flag>true</request_flag>
  </Resource>
</Topology>
```

```
<Topology>   <!-- OL1 - extended customer request -->
  <!-- H-S=L=S-H -->
  <GraphLabel>
    <!-- we filled in gaps in the original customer request -->
    <id>p:c1n1o1gl1</id>
    <role_identifier>PIP1</role_identifier>
    <graph_type>OL</graph_type>
    <graph_nr>1</graph_nr>
    <net_identifier>VNet1</net_identifier>
  </GraphLabel>
  <NetworkElement>
    <id>p:c1n1o1h1</id>
    <identifier>VHost1</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:c1n1o1gl1</graphLabel>
    <hostedNetworkElement>p:c1n1o0h1</hostedNetworkElement>
    <hostingNetworkElement>p:c1n1m0h1</hostingNetworkElement>
    <resource>p:c1n1o1h1ram</resource>
    <resource>p:c1n1o1h1hd</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country</attribute>
        <value>/NA/USA</value>
        <priority>1</priority>
        <request_flag>true</request_flag>
      </Feature>
      <Feature>
        <attribute>/position/continent/country</attribute>
        <value>/NA/Canada</value>
        <priority>2</priority>
        <request_flag>true</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1o1h1ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o1l1s1i0</connectedInterface>
        <hostedNetworkInterface>p:c1n1o0h1ieth0</hostedNetworkInterface>
        <hostingNetworkInterface>p:c1n1m0h1ieth0</hostingNetworkInterface>
        <resource>p:c1n1o1h1ieth0nbw</resource>
        <resource>p:c1n1o1h1ieth0lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:c1n1o1h2</id>
    <identifier>VHost2</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:c1n1o1gl1</graphLabel>
    <hostedNetworkElement>p:c1n1o0h2</hostedNetworkElement>
    <hostingNetworkElement>p:c1n1m0h2</hostingNetworkElement>
    <resource>p:c1n1o1h2ram</resource>
```

```
  <resource>p:c1n1o1h2hd</resource>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1o1h2ieth0</id>
      <identifier>eth0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1o1l1s2i0</connectedInterface>
      <hostedNetworkInterface>p:c1n1o0h2ieth0</hostedNetworkInterface>
      <hostingNetworkInterface>p:c1n1m0h2ieth0</hostingNetworkInterface>
      <resource>p:c1n1o1h2ieth0nbw</resource>
      <resource>p:c1n1o1h2ieth0lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:c1n1o1l1s1</id>
  <identifier>VH1VH2Link Splitter 1</identifier>
  <ne_type>/node/splitter/generic</ne_type>
  <graphLabel>p:c1n1o2gl1</graphLabel>
  <hostingNetworkElement>p:c1n1m0l1s1</hostingNetworkElement>
  <resource>p:c1n1o1l1s1bw</resource>
  <!-- allowing for 0.5ms latency in splitter -->
  <resource>p:c1n1o1l1s1lat</resource>
  <features>
    <Feature>
      <attribute>/not/position/continent/country</attribute>
      <value>/Asia/Afghanistan</value>
      <priority>1</priority>
      <request_flag>true</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1o1l1s1i0</id>
      <identifier>VH1VH2Link Splitter 1 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1o1h1ieth0</connectedInterface>
      <hostingNetworkInterface>p:c1n1m0l1s1i0</hostingNetworkInterface>
      <resource>p:c1n1o1l1s1i0nbw</resource>
      <resource>p:c1n1o1l1s1i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1o1l1s1i1</id>
      <identifier>VH1VH2Link Splitter 1 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1o1l1p1i0</connectedInterface>
      <hostingNetworkInterface>p:c1n1m0l1s1i1</hostingNetworkInterface>
      <resource>p:c1n1o1l1s1i1nbw</resource>
      <resource>p:c1n1o1l1s1i1lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1o1l1s1i2</id>
      <identifier>VH1VH2Link Splitter 1 Interface 2</identifier>
      <ni_type>/interface/generic</ni_type>
```

```
        <connectedInterface>p:c1n1o1l1p2i0</connectedInterface>
        <hostingNetworkInterface>p:c1n1m0l1s1i2</hostingNetworkInterface>
        <resource>p:c1n1o1l1s1i2nbw</resource>
        <resource>p:c1n1o1l1s1i2lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:c1n1o1l1p1</id>
    <identifier>VH1VH2Link Path1</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:c1n1o1gl1</graphLabel>
    <hostingNetworkElement>p:c1n1m0l1p1p1</hostingNetworkElement>
    <hostingNetworkElement>p:c1n1m0l1p1p2</hostingNetworkElement>
    <hostingNetworkElement>p:c1n1m0l1p1p3</hostingNetworkElement>
    <!-- we need 100 MBit/s full-duplex -->
    <resource>p:c1n1o1l1p1ubw</resource>
    <resource>p:c1n1o1l1p1dbw</resource>
    <!-- we determined that latency may not exceed 10 ms -->
    <resource>p:c1n1o1l1p1lat</resource>
    <features>
      <Feature>
        <attribute>/not/position/continent/country</attribute>
        <value>/Asia/Afghanistan</value>
        <priority>1</priority>
        <request_flag>true</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1o1l1p1i0</id>
        <identifier>VH1VH2Link Path 1 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o1l1s1i1</connectedInterface>
        <hostingNetworkInterface>p:c1n1m0l1p1p1i0</hostingNetworkInterface>
        <resource>p:c1n1o1l1p1i0nbw</resource>
        <resource>p:c1n1o1l1p1i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:c1n1o1l1p1i1</id>
        <identifier>VH1VH2Link Path 1 Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o1l1s2i1</connectedInterface>
        <hostingNetworkInterface>p:c1n1m0l1p1p3i1</hostingNetworkInterface>
        <resource>p:c1n1o1l1p1i1nbw</resource>
        <resource>p:c1n1o1l1p1i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:c1n1o1l1p2</id>
    <identifier>VH1VH2Link Path2</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:c1n1o1gl1</graphLabel>
```

```
  <hostingNetworkElement>p:c1n1m0l1p2p1</hostingNetworkElement>
  <hostingNetworkElement>p:c1n1m0l1p2p2</hostingNetworkElement>
  <hostingNetworkElement>p:c1n1m0l1p2p3</hostingNetworkElement>
  <!-- we need 100 MBit/s full-duplex -->
  <resource>p:c1n1o1l1p2ubw</resource>
  <resource>p:c1n1o1l1p2dbw</resource>
  <!-- we determined that latency may not exceed 10 ms -->
  <resource>p:c1n1o1l1p2lat</resource>
  <features>
    <Feature>
      <attribute>/not/position/continent/country</attribute>
      <value>/Asia/Afghanistan</value>
      <priority>1</priority>
      <request_flag>true</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1o1l1p2i0</id>
      <identifier>VH1VH2Link Path 2 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1o1l1s1i2</connectedInterface>
      <hostingNetworkInterface>p:c1n1m0l1p2p1i0</hostingNetworkInterface>
      <resource>p:c1n1o1l1p2i0nbw</resource>
      <resource>p:c1n1o1l1p2i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1o1l1p2i1</id>
      <identifier>VH1VH2Link Path 2 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1o1l1s2i2</connectedInterface>
      <hostingNetworkInterface>p:c1n1m0l1p2p3i1</hostingNetworkInterface>
      <resource>p:c1n1o1l1p2i1nbw</resource>
      <resource>p:c1n1o1l1p2i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:c1n1o1l1s2</id>
  <identifier>VH1VH2Link Splitter 2</identifier>
  <ne_type>/node/splitter/generic</ne_type>
  <graphLabel>p:c1n1o2gl1</graphLabel>
  <hostingNetworkElement>p:c1n1m0l1s2</hostingNetworkElement>
  <resource>p:c1n1o1l1s2bw</resource>
  <!-- allowing for 0.5ms latency in splitter -->
  <resource>p:c1n1o1l1s2lat</resource>
  <features>
    <Feature>
      <attribute>/not/position/continent/country</attribute>
      <value>/Asia/Afghanistan</value>
      <priority>1</priority>
      <request_flag>true</request_flag>
    </Feature>
  </features>
```

```
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1o1l1s2i0</id>
        <identifier>VH1VH2Link Splitter 2 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o1h2ieth0</connectedInterface>
        <hostingNetworkInterface>p:c1n1m0l1s2i0</hostingNetworkInterface>
        <resource>p:c1n1o1l1s2i0nbw</resource>
        <resource>p:c1n1o1l1s2i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:c1n1o1l1s2i1</id>
        <identifier>VH1VH2Link Splitter 2 Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o1l1p1i1</connectedInterface>
        <hostingNetworkInterface>p:c1n1m0l1s2i1</hostingNetworkInterface>
        <resource>p:c1n1o1l1s2i1nbw</resource>
        <resource>p:c1n1o1l1s2i1lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:c1n1o1l1s2i2</id>
        <identifier>VH1VH2Link Splitter 2 Interface 2</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1o1l1p2i1</connectedInterface>
        <hostingNetworkInterface>p:c1n1m0l1s2i2</hostingNetworkInterface>
        <resource>p:c1n1o1l1s2i2nbw</resource>
        <resource>p:c1n1o1l1s2i2lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <!-- Resources -->
  <Resource>
    <id>p:c1n1o1h1ram</id>
    <attribute>/host/generic/RAM/real/amount</attribute>
    <value>1</value>
    <resource_unit>GB</resource_unit>
    <value_type>minimum</value_type>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1h1hd</id>
    <attribute>/host/generic/HDD/space</attribute>
    <value>10</value>
    <resource_unit>GB</resource_unit>
    <value_type>minimum</value_type>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1h2ram</id>
    <attribute>/host/generic/RAM/real/amount</attribute>
    <value>1</value>
    <resource_unit>GB</resource_unit>
    <value_type>minimum</value_type>
    <request_flag>true</request_flag>
```

```
</Resource>
<Resource>
  <id>p:c1n1o1h2hd</id>
  <attribute>/host/generic/HDD/space</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1bw</id>
  <attribute>/node/splitter/generic/forward/bandwidth</attribute>
  <!-- 100 per direction -->
  <value>200</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1lat</id>
  <attribute>/node/splitter/generic/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <value_type>maximum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1ubw</id>
  <!-- up-/down:
    - full duplex links not shared,
    - hence exactly two interfaces
    - prototype uses convention on interface order
    - could also be reflected in ids/identifier/attributes $foo  -->
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <composing_operation>minimum</composing_operation>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <composing_operation>minimum</composing_operation>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>9</value>
  <resource_unit>ms</resource_unit>
```

```
    <value_type>maximum</value_type>
    <composing_operation>sum</composing_operation>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1l1p2ubw</id>
    <!-- up-/down:
      - full duplex links not shared,
      - hence exactly two interfaces
      - prototype uses convention on interface order
      - could also be reflected in ids/identifier/attributes $foo  -->
    <attribute>/link/generic/upstream/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <value_type>minimum</value_type>
    <composing_operation>minimum</composing_operation>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1l1p2dbw</id>
    <attribute>/link/generic/downstream/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <value_type>minimum</value_type>
    <composing_operation>minimum</composing_operation>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1l1p2lat</id>
    <attribute>/link/generic/latency</attribute>
    <value>9</value>
    <resource_unit>ms</resource_unit>
    <value_type>maximum</value_type>
    <composing_operation>sum</composing_operation>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1l1s2bw</id>
    <attribute>/node/splitter/generic/forward/bandwidth</attribute>
    <!-- 100 per direction -->
    <value>200</value>
    <resource_unit>Mbps</resource_unit>
    <value_type>minimum</value_type>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1o1l1s2lat</id>
    <attribute>/node/splitter/generic/forward/latency</attribute>
    <value>0.5</value>
    <resource_unit>ms</resource_unit>
    <value_type>maximum</value_type>
    <request_flag>true</request_flag>
  </Resource>
  <Resource>
```

```
  <id>p:c1n1o1h1ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1h1ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1h2ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1h2ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
```

```
</Resource>
<Resource>
  <id>p:c1n1o1l1s1i1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1i2nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s1i2lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
```

```
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p2i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p2i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1p2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s2i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <value_type>minimum</value_type>
  <request_flag>true</request_flag>
</Resource>
<Resource>
  <id>p:c1n1o1l1s2i1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
```

```
      <value>100</value>
      <resource_unit>Mbps</resource_unit>
      <value_type>minimum</value_type>
      <request_flag>true</request_flag>
    </Resource>
    <Resource>
      <id>p:c1n1o1l1s2i2nbw</id>
      <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
      <value>100</value>
      <resource_unit>Mbps</resource_unit>
      <value_type>minimum</value_type>
      <request_flag>true</request_flag>
    </Resource>
    <Resource>
      <id>p:c1n1o1l1s2i2lbw</id>
      <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
      <value>100</value>
      <resource_unit>Mbps</resource_unit>
      <value_type>minimum</value_type>
      <request_flag>true</request_flag>
    </Resource>
    <!-- ConstraintGroups -->
    <ConstraintGroup>
      <networkElement>p:c1n1o1h1</networkElement>
      <networkElement>p:c1n1o1l1s1</networkElement>
      <Constraint>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <relation>equals</relation>
        <priority>1</priority>
      </Constraint>
    </ConstraintGroup>
    <ConstraintGroup>
      <networkElement>p:c1n1o1h2</networkElement>
      <networkElement>p:c1n1o1l1s2</networkElement>
      <Constraint>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <relation>equals</relation>
        <priority>1</priority>
      </Constraint>
    </ConstraintGroup>
    <ConstraintGroup>
      <networkElement>p:c1n1o1l1p1</networkElement>
      <networkElement>p:c1n1o1l1p2</networkElement>
      <Constraint>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <relation>distinct</relation>
        <priority>1</priority>
      </Constraint>
    </ConstraintGroup>
  </Topology>


  <Topology>
```

```
<!-- ML - mapping request to substrate and annotating set Features etc. -->
<!-- not repeating request ConstraintGroups, Resources and Features
  - for the sake of readability-->
<!-- H-S=L=L=L=S-H -->
<GraphLabel>
  <id>p:c1n1m0gl1</id>
  <role_identifier>PIP1</role_identifier>
  <graph_type>ML</graph_type>
  <graph_nr>0</graph_nr>
  <net_identifier>VNet1</net_identifier>
</GraphLabel>
<NetworkElement>
  <id>p:c1n1m0h1</id>
  <identifier>VHost1</identifier>
  <ne_type>/node/host/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1h1</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h1</hostingNetworkElement>
  <resource>p:c1n1m0h1ram</resource>
  <resource>p:c1n1m0h1hd</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country</attribute>
      <value>/NA/USA</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1m0h1ieth0</id>
      <identifier>eth0</identifier>
      <!-- as an example, let's use the alias field
        - to convey the name of the virtual interface
        - that it has on the substrate host -->
      <alias>vifh0i0</alias>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1s1i0</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1h1ieth0</hostedNetworkInterface>
      <resource>p:c1n1m0h1ieth0nbw</resource>
      <resource>p:c1n1m0h1ieth0lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
  <customerConsoleInterface>p:cci1</customerConsoleInterface>
  <providerConsoleInterface>p:pci1</providerConsoleInterface>
</NetworkElement>
<NetworkElement>
  <id>p:c1n1m0h2</id>
  <identifier>VHost2</identifier>
  <ne_type>/node/host/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1h2</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h2</hostingNetworkElement>
  <resource>p:c1n1m0h2ram</resource>
```

```
    <resource>p:c1n1m0h2hd</resource>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1m0h2ieth0</id>
        <identifier>eth0</identifier>
        <alias>vifh1i0</alias>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1m0l1s2i0</connectedInterface>
        <hostedNetworkInterface>p:c1n1o1h2ieth0</hostedNetworkInterface>
        <resource>p:c1n1m0h2ieth0nbw</resource>
        <resource>p:c1n1m0h2ieth0lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
    <customerConsoleInterface>p:cci1</customerConsoleInterface>
    <providerConsoleInterface>p:pci2</providerConsoleInterface>
  </NetworkElement>
  <NetworkElement>
    <id>p:c1n1m0l1s1</id>
    <identifier>VH1VH2Link Splitter 1</identifier>
    <ne_type>/node/splitter/generic</ne_type>
    <graphLabel>p:c1n1m0gl1</graphLabel>
    <hostedNetworkElement>p:c1n1o1l1s1</hostedNetworkElement>
    <hostingNetworkElement>p:p1n1u1h1</hostingNetworkElement>
    <resource>p:c1n1m0l1s1bw</resource>
    <!-- allowing for 0.5ms latency in splitter -->
    <resource>p:c1n1m0l1s1lat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country</attribute>
        <value>/NA/USA</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1m0l1s1i0</id>
        <identifier>VH1VH2Link Splitter 1 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1m0h1ieth0</connectedInterface>
        <hostedNetworkInterface>p:c1n1o1l1s1i0</hostedNetworkInterface>
        <resource>p:c1n1m0l1s1i0nbw</resource>
        <resource>p:c1n1m0l1s1i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:c1n1m0l1s1i1</id>
        <identifier>VH1VH2Link Splitter 1 Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1m0l1p1p1i0</connectedInterface>
        <hostedNetworkInterface>p:c1n1o1l1s1i1</hostedNetworkInterface>
        <resource>p:c1n1m0l1s1i1nbw</resource>
        <resource>p:c1n1m0l1s1i1lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
```

```
      <id>p:c1n1m0l1s1i2</id>
      <identifier>VH1VH2Link Splitter 1 Interface 2</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p2p1i0</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1l1s1i2</hostedNetworkInterface>
      <resource>p:c1n1m0l1s1i2nbw</resource>
      <resource>p:c1n1m0l1s1i2lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
  <!-- This component is transparent to the customer -->
  <providerConsoleInterface>p:pci1</providerConsoleInterface>
</NetworkElement>
<NetworkElement>
  <!-- We'll have the first path cross a tunnel (logical 1 hop)
    - It is hosted in part on the end hosts, and in part on the tunnel link
    - thus split in 3 parts -->
  <id>p:c1n1m0l1p1p1</id>
  <identifier>VH1VH2Link Path1 Part1</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1l1p1</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h1</hostingNetworkElement>
  <resource>p:c1n1m0l1p1p1ubw</resource>
  <resource>p:c1n1m0l1p1p1dbw</resource>
  <!-- this segment may use max. 0.5 ms -->
  <resource>p:c1n1m0l1p1p1lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country</attribute>
      <value>/NA/USA</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1m0l1p1p1i0</id>
      <identifier>VH1VH2Link Path 1 Part 1 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1s1i1</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1l1p1i0</hostedNetworkInterface>
      <resource>p:c1n1m0l1p1p1i0nbw</resource>
      <resource>p:c1n1m0l1p1p1i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1p1p1i1</id>
      <identifier>VH1VH2Link Path 1 Part 1 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p1p2i0</connectedInterface>
      <hostingNetworkInterface>p:p1n1u1h1ieth0</hostingNetworkInterface>
      <resource>p:c1n1m0l1p1p1i1nbw</resource>
      <resource>p:c1n1m0l1p1p1i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
```

```
</NetworkElement>
<NetworkElement>
  <id>p:c1n1m0l1p1p2</id>
  <identifier>VH1VH2Link Path1 Part2</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1l1p1</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1l1</hostingNetworkElement>
  <resource>p:c1n1m0l1p1p2ubw</resource>
  <resource>p:c1n1m0l1p1p2dbw</resource>
  <!-- this segment may use max. 8 ms -->
  <resource>p:c1n1m0l1p1p2lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country</attribute>
      <!-- This one crosses the border
        - it's going across both countries -->
      <value>/NA/USA:/NA/Canada</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1m0l1p1p2i0</id>
      <identifier>VH1VH2Link Path 1 Part 2 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p1p1i1</connectedInterface>
      <hostingNetworkInterface>p:p1n1u1l1i0</hostingNetworkInterface>
      <resource>p:c1n1m0l1p1p2i0nbw</resource>
      <resource>p:c1n1m0l1p1p2i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1p1p2i1</id>
      <identifier>VH1VH2Link Path 1 Part 2 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p1p3i0</connectedInterface>
      <hostingNetworkInterface>p:p1n1u1l1i1</hostingNetworkInterface>
      <resource>p:c1n1m0l1p1p2i1nbw</resource>
      <resource>p:c1n1m0l1p1p2i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:c1n1m0l1p1p3</id>
  <identifier>VH1VH2Link Path1 Part3</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1l1p1</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h2</hostingNetworkElement>
  <resource>p:c1n1m0l1p1p3ubw</resource>
  <resource>p:c1n1m0l1p1p3dbw</resource>
  <!-- this segment may use max. 8 ms -->
  <resource>p:c1n1m0l1p1p3lat</resource>
```

```
  <features>
    <Feature>
      <attribute>/position/continent/country</attribute>
      <!-- This one crosses the border
        - it's going across both countries -->
      <value>/NA/Canada</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1m0l1p1p3i0</id>
      <identifier>VH1VH2Link Path 1 Part 3 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p1p2i1</connectedInterface>
      <hostingNetworkInterface>p:p1n1u1h2ieth0</hostingNetworkInterface>
      <resource>p:c1n1m0l1p1p3i0nbw</resource>
      <resource>p:c1n1m0l1p1p3i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1p1p3i1</id>
      <identifier>VH1VH2Link Path 1 Part 3 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1s2i1</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1l1p1i1</hostedNetworkInterface>
      <resource>p:c1n1m0l1p1p3i1nbw</resource>
      <resource>p:c1n1m0l1p1p3i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <!-- The second path runs via a true 1 hop
    - It is hosted in part on the end hosts, and in part on the link
    - thus split in 3 parts, too
    - not to be differentiated on ML level, but will be visible in UL -->
  <id>p:c1n1m0l1p2p1</id>
  <identifier>VH1VH2Link Path2 Part1</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1l1p2</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h1</hostingNetworkElement>
  <resource>p:c1n1m0l1p2p1ubw</resource>
  <resource>p:c1n1m0l1p2p1dbw</resource>
  <!-- this segment may use max. 0.5 ms -->
  <resource>p:c1n1m0l1p2p1lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country</attribute>
      <value>/NA/USA</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
```

```
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1m0l1p2p1i0</id>
        <identifier>VH1VH2Link Path 2 Part 1 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1m0l1s1i2</connectedInterface>
        <hostedNetworkInterface>p:c1n1o1l1p2i0</hostedNetworkInterface>
        <resource>p:c1n1m0l1p2p1i0nbw</resource>
        <resource>p:c1n1m0l1p2p1i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:c1n1m0l1p2p1i1</id>
        <identifier>VH1VH2Link Path 2 Part 1 Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1m0l1p2p2i0</connectedInterface>
        <hostingNetworkInterface>p:p1n1u1h1itun0</hostingNetworkInterface>
        <resource>p:c1n1m0l1p2p1i1nbw</resource>
        <resource>p:c1n1m0l1p2p1i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:c1n1m0l1p2p2</id>
    <identifier>VH1VH2Link Path2 Part2</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:c1n1m0gl1</graphLabel>
    <hostedNetworkElement>p:c1n1o1l1p2</hostedNetworkElement>
    <hostingNetworkElement>p:p1n1u1l5p1</hostingNetworkElement>
    <!-- We could define individual parts for the specific parts of l5
      - but as we don't configure anything (e.g. interfaces) on these parts,
      - we may as well keep it simple (and the example shorter) -->
    <hostingNetworkElement>p:p1n1u1l5p2</hostingNetworkElement>
    <hostingNetworkElement>p:p1n1u1l5p3</hostingNetworkElement>
    <resource>p:c1n1m0l1p2p2ubw</resource>
    <resource>p:c1n1m0l1p2p2dbw</resource>
    <!-- this segment may use max. 8 ms -->
    <resource>p:c1n1m0l1p2p2lat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country</attribute>
        <value>/NA/USA:/NA/Canada</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:c1n1m0l1p2p2i0</id>
        <identifier>VH1VH2Link Path 2 Part 2 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:c1n1m0l1p2p1i1</connectedInterface>
        <hostingNetworkInterface>p:p1n1u1l5p1i0</hostingNetworkInterface>
        <resource>p:c1n1m0l1p2p2i0nbw</resource>
        <resource>p:c1n1m0l1p2p2i0lbw</resource>
```

```
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1p2p2i1</id>
      <identifier>VH1VH2Link Path 2 Part 2 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p2p3i0</connectedInterface>
      <hostingNetworkInterface>p:p1n1u1l5p3i1</hostingNetworkInterface>
      <resource>p:c1n1m0l1p2p2i1nbw</resource>
      <resource>p:c1n1m0l1p2p2i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:c1n1m0l1p2p3</id>
  <identifier>VH1VH2Link Path2 Part3</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1l1p2</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h2</hostingNetworkElement>
  <resource>p:c1n1m0l1p2p3ubw</resource>
  <resource>p:c1n1m0l1p2p3dbw</resource>
  <!-- this segment may use max. 8 ms -->
  <resource>p:c1n1m0l1p2p3lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country</attribute>
      <value>/NA/Canada</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1m0l1p2p3i0</id>
      <identifier>VH1VH2Link Path 2 Part 3 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p2p2i1</connectedInterface>
      <!-- without interface mapping,
        - the corresponding resource assignment would be ambiguous
        - which would not be a problem in our situation
        - in which PIP2 and PIP3 do not directly interconnect
        - in the substrate description
        - (could be inferred by looking at other link end)
        - but might be, if they were
        - (2 paths between SHost1 and SHost2) -->
      <hostingNetworkInterface>p:p1n1u1h2itun0</hostingNetworkInterface>
      <resource>p:c1n1m0l1p2p3i0nbw</resource>
      <resource>p:c1n1m0l1p2p3i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1p2p3i1</id>
      <identifier>VH1VH2Link Path 2 Part 3 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1s2i2</connectedInterface>
```

```
      <hostedNetworkInterface>p:c1n1o1l1p2i1</hostedNetworkInterface>
      <resource>p:c1n1m0l1p2p3i1nbw</resource>
      <resource>p:c1n1m0l1p2p3i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:c1n1m0l1s2</id>
  <identifier>VH1VH2Link Splitter 2</identifier>
  <ne_type>/node/splitter/generic</ne_type>
  <graphLabel>p:c1n1m0gl1</graphLabel>
  <hostedNetworkElement>p:c1n1o1l1s2</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u1h2</hostingNetworkElement>
  <resource>p:c1n1m0l1s2bw</resource>
  <!-- allowing for 0.5ms latency in splitter -->
  <resource>p:c1n1m0l1s2lat</resource>
  <features>
    <Feature>
      <attribute>/not/position/continent/country</attribute>
      <value>/Asia/Afghanistan</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:c1n1m0l1s2i0</id>
      <identifier>VH1VH2Link Splitter 2 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0h2ieth0</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1l1s2i0</hostedNetworkInterface>
      <resource>p:c1n1m0l1s2i0nbw</resource>
      <resource>p:c1n1m0l1s2i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1s2i1</id>
      <identifier>VH1VH2Link Splitter 2 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p1p3i1</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1l1s2i1</hostedNetworkInterface>
      <resource>p:c1n1m0l1s2i1nbw</resource>
      <resource>p:c1n1m0l1s2i1lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:c1n1m0l1s2i2</id>
      <identifier>VH1VH2Link Splitter 2 Interface 2</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:c1n1m0l1p2p3i1</connectedInterface>
      <hostedNetworkInterface>p:c1n1o1l1s2i2</hostedNetworkInterface>
      <resource>p:c1n1m0l1s2i2nbw</resource>
      <resource>p:c1n1m0l1s2i2lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
  <!-- This component is transparent to the customer -->
```

```
  <providerConsoleInterface>p:pci2</providerConsoleInterface>
</NetworkElement>
<!-- Resources -->
<Resource>
  <id>p:c1n1m0h1ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>1</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0h1hd</id>
  <attribute>/host/generic/HDD/space</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0h2ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>1</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0h2hd</id>
  <attribute>/host/generic/HDD/space</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1bw</id>
  <attribute>/node/splitter/generic/forward/bandwidth</attribute>
  <!-- 100 per direction -->
  <value>200</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1lat</id>
  <attribute>/node/splitter/generic/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p1ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
```

```
  <id>p:c1n1m0l1p1p1dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p1lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>8</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
```

```
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>8</value>
  <resource_unit>ms</resource_unit>
  <composing_operation>sum</composing_operation>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p3ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p3dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
```

```
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0l1p2p3lat</id>
    <attribute>/link/generic/latency</attribute>
    <value>0.5</value>
    <resource_unit>ms</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0l1s2bw</id>
    <attribute>/node/splitter/generic/forward/bandwidth</attribute>
    <value>200</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0l1s2lat</id>
    <attribute>/node/splitter/generic/forward/latency</attribute>
    <value>0.5</value>
    <resource_unit>ms</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0h1ieth0nbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0h1ieth0lbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0h2ieth0nbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0h2ieth0lbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
```

```
  <id>p:c1n1m0l1s1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1i1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1i2nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s1i2lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
```

```
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
```

```
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p1p3i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2i0lbw</id>
```

```
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p3i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p3i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p3i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1p2p3i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:c1n1m0l1s2i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>100</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
```

```
  <Resource>
    <id>p:c1n1m0l1s2i1nbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0l1s2i1lbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0l1s2i2nbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:c1n1m0l1s2i2lbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>100</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
</Topology>



<Topology>   <!-- UL1 - describing logical structure of substrate -->
  <!-- did not model switches and routers to keep example short(er) -->
  <!-- H3-L3-H1=L1/5=H2 ; H1-L2-P2(...unknown...)P3-L4-H2 -->
  <GraphLabel>
    <!-- modeled NFS resource as shared resource, described tunnel -->
    <id>p:p1n1u1gl1</id>
    <role_identifier>PIP1</role_identifier>
    <graph_type>UL</graph_type>
    <graph_nr>1</graph_nr>
  </GraphLabel>
  <NetworkElement>
    <id>p:p1n1u1h1</id>
    <identifier>SHost1</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:p1n1u1gl1</graphLabel>
    <hostedNetworkElement>p:c1n1m0h1</hostedNetworkElement>
    <hostedNetworkElement>p:c1n1m0l1s1</hostedNetworkElement>
    <hostedNetworkElement>p:c1n1m0l1p1p1</hostedNetworkElement>
    <hostedNetworkElement>p:c1n1m0l1p2p1</hostedNetworkElement>
    <hostingNetworkElement>p:p1n1u0h1</hostingNetworkElement>
    <resource>p:p1n1u1h1ram</resource>
    <resource>p:p1n1u1h1hd1</resource>
```

```
<resource>p:p1n1u1nfs1</resource>
<resource>p:p1n1u1h1sbw</resource>
<resource>p:p1n1u1h1slat</resource>
<resource>p:p1n1u1h1fbw</resource>
<resource>p:p1n1u1h1flat</resource>
<features>
  <Feature>
    <attribute>/position/continent/country/state/city/site/device</attribute>
    <value>/NA/USA/MA/Boston/somesite/SHost1</value>
    <priority>1</priority>
    <request_flag>false</request_flag>
  </Feature>
</features>
<networkInterfaces>
  <!-- direct link to SHost2 in Canada -->
  <NetworkInterface>
    <id>p:p1n1u1h1ieth0</id>
    <identifier>eth0</identifier>
    <ni_type>/interface/generic</ni_type>
    <connectedInterface>p:p1n1u1l1i0</connectedInterface>
    <hostedNetworkInterface>p:c1n1m0l1p1p1i1</hostedNetworkInterface>
    <hostingNetworkInterface>p:p1n1u0h1ieth0</hostingNetworkInterface>
    <resource>p:p1n1u1h1ieth0nbw</resource>
    <resource>p:p1n1u1h1ieth0lbw</resource>
  </NetworkInterface>
  <!-- physical link to neighbour Provider PIP2 -->
  <NetworkInterface>
    <id>p:p1n1u1h1ieth1</id>
    <identifier>eth1</identifier>
    <ni_type>/interface/generic</ni_type>
    <connectedInterface>p:p1n1u1l2i0</connectedInterface>
    <hostingNetworkInterface>p:p1n1u0h1ieth1</hostingNetworkInterface>
    <!-- it shares its bandwidth with tun0
      - partial in one direction! -->
    <resource>p:p1n1u1h1ieth1snbw</resource>
    <resource>p:p1n1u1h1ieth1rnbw</resource>
    <resource>p:p1n1u1h1ieth1lbw</resource>
  </NetworkInterface>
  <!-- physical link to unused SHost3 which shares NFS volume -->
  <NetworkInterface>
    <id>p:p1n1u1h1ieth2</id>
    <identifier>eth2</identifier>
    <ni_type>/interface/generic</ni_type>
    <connectedInterface>p:p1n1u1l3i0</connectedInterface>
    <hostingNetworkInterface>p:p1n1u0h1ieth2</hostingNetworkInterface>
    <resource>p:p1n1u1h1ieth2nbw</resource>
    <resource>p:p1n1u1h1ieth2lbw</resource>
  </NetworkInterface>
  <!-- tunnel link to SHost2 via PIP2, some not-known-network and PIP3 -->
  <NetworkInterface>
    <id>p:p1n1u1h1itun0</id>
    <identifier>tun0</identifier>
    <ni_type>/interface/generic</ni_type>
    <connectedInterface>p:p1n1u1l5p1i0</connectedInterface>
```

```
      <hostedNetworkInterface>p:c1n1m0l1p2p1i1</hostedNetworkInterface>
      <hostingNetworkInterface>p:p1n1u0h1ieth1</hostingNetworkInterface>
      <!-- it shares its bandwidth with eth1
        - partial in one direction! -->
      <resource>p:p1n1u1h1ieth1snbw</resource>
      <resource>p:p1n1u1h1ieth1rnbw</resource>
      <resource>p:p1n1u1h1ieth1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u1h2</id>
  <identifier>SHost2</identifier>
  <ne_type>/node/host/generic</ne_type>
  <graphLabel>p:p1n1u1gl1</graphLabel>
  <hostedNetworkElement>p:c1n1m0h2</hostedNetworkElement>
  <hostedNetworkElement>p:c1n1m0l1s2</hostedNetworkElement>
  <hostedNetworkElement>p:c1n1m0l1p1p3</hostedNetworkElement>
  <hostedNetworkElement>p:c1n1m0l1p2p3</hostedNetworkElement>
  <hostingNetworkElement>p:p1n1u0h2</hostingNetworkElement>
  <resource>p:p1n1u1h2ram</resource>
  <resource>p:p1n1u1h2hd</resource>
  <resource>p:p1n1u1h1sbw</resource>
  <resource>p:p1n1u1h1slat</resource>
  <resource>p:p1n1u1h1fbw</resource>
  <resource>p:p1n1u1h1flat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/Canada/QC/Montreal/somesite/SHost2</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <!-- direct link to SHost 1 -->
    <NetworkInterface>
      <id>p:p1n1u1h2ieth0</id>
      <identifier>eth0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1l1i1</connectedInterface>
      <hostedNetworkInterface>p:c1n1m0l1p1p3i0</hostedNetworkInterface>
      <hostingNetworkInterface>p:p1n1u0h2ieth0</hostingNetworkInterface>
      <resource>p:p1n1u1l1h2ieth0nbw</resource>
      <resource>p:p1n1u1l1h2ieth0lbw</resource>
    </NetworkInterface>
    <!-- physical link to neighbour Provider PIP3 -->
    <NetworkInterface>
      <id>p:p1n1u1h2ieth1</id>
      <identifier>eth1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1l4i0</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0h2ieth1</hostingNetworkInterface>
      <resource>p:p1n1u1l1h2ieth1nbw</resource>
```

```
        <resource>p:p1n1u1l1h2ieth1lbw</resource>
      </NetworkInterface>
      <!-- tunnel link to SHost1 via PIP3, some not-known-network and PIP2 -->
      <NetworkInterface>
        <id>p:p1n1u1h2itun0</id>
        <identifier>tun0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1l5p3i1</connectedInterface>
        <hostedNetworkInterface>p:c1n1m0l1p2p3i0</hostedNetworkInterface>
        <hostingNetworkInterface>p:p1n1u0h2ieth1</hostingNetworkInterface>
        <resource>p:p1n1u1h2ieth1nbw</resource>
        <resource>p:p1n1u1h2ieth1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u1h3</id>
    <identifier>SHost3</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:p1n1u1gl1</graphLabel>
    <hostingNetworkElement>p:p1n1u0h3</hostingNetworkElement>
    <resource>p:p1n1u1h3ram</resource>
    <resource>p:p1n1u1nfs1</resource>
    <resource>p:p1n1u1h1fbw</resource>
    <resource>p:p1n1u1h1flat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SHost3</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <!-- direct link to SHost 1 -->
      <NetworkInterface>
        <id>p:p1n1u1h3ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1l3i1</connectedInterface>
        <hostingNetworkInterface>p:p1n1u0h3ieth0</hostingNetworkInterface>
        <resource>p:p1n1u1h3ieth0nbw</resource>
        <resource>p:p1n1u1h3ieth0lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <!-- This is a neighbouring PIP2 we have an uplink to
      - it's interconneted with PIP3,
      - via which we have an alternative tunnel link to SHost2 -->
    <id>p:p1n1u1p2</id>
    <identifier>PIP2</identifier>
    <ne_type>/node/pip/generic</ne_type>
    <graphLabel>p:p1n1u1gl1</graphLabel>
```

```
  <hostingNetworkElement>p:p1n1u0p2</hostingNetworkElement>
  <!-- we don't have authority to host things there
    - we could model forwarding resources we leased for the tunnel
    - and then split L5P2 below into two parts
    - (with respective negotiated latency parts)
    - the latency parts would then reflect our leasing SLA with PIP2 and PIP3
    - (could also be full for PIP2 and 0 for PIP3, if PIP2 acted as reseller)
    - but we'll omit this and not map L5P2 to shorten the example -->
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city</attribute>
      <value>/NA/USA/MA/Boston</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <!-- direct link to SHost 1 -->
    <NetworkInterface>
      <id>p:p1n1u1p2ieth0</id>
      <identifier>eth0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1l2i1</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0p2ieth0</hostingNetworkInterface>
      <resource>p:p1n1u1p2ieth0nbw</resource>
      <resource>p:p1n1u1p2ieth0lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <!-- This is a neighbouring PIP3 we have an uplink to
    - it's interconneted with PIP2,
    - via which we have an alternative tunnel link to SHost1 -->
  <id>p:p1n1u1p3</id>
  <identifier>PIP3</identifier>
  <ne_type>/node/pip/generic</ne_type>
  <graphLabel>p:p1n1u1gl1</graphLabel>
  <hostingNetworkElement>p:p1n1u0p3</hostingNetworkElement>
  <!-- we don't have authority to host things there
    - we could model substrate forwarding resources we leased for the tunnel
    - and then split L5P2 below into two parts
    - (with respective negotiated latency parts)
    - the latency parts would then reflect our leasing SLA with PIP2 and PIP3
    - (could also be full for PIP2 and 0 for PIP3, if PIP2 acted as reseller)
    - but we'll omit this and not map L5P2 to shorten the example -->
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city</attribute>
      <value>/NA/Canada/QC/Montreal</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
```

```
      <!-- direct link to SHost 2 -->
      <NetworkInterface>
        <id>p:p1n1u1p3ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1l4i1</connectedInterface>
        <hostingNetworkInterface>p:p1n1u0p3ieth0</hostingNetworkInterface>
        <resource>p:p1n1u1p3ieth0nbw</resource>
        <resource>p:p1n1u1p3ieth0lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u1l1</id>
    <identifier>SH1SH2Link</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:p1n1u1gl1</graphLabel>
    <hostedNetworkElement>p:c1n1m0l1p2</hostedNetworkElement>
    <hostingNetworkElement>p:p1n1u0l1</hostingNetworkElement>
    <resource>p:p1n1u1l1ubw</resource>
    <resource>p:p1n1u1l1dbw</resource>
    <resource>p:p1n1u1l1lat</resource>
    <features>
      <Feature>
        <!-- Note: in principle, one might list all positions in here
          - to verify position compliance
          - omitted for sake of simplicity -->
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SH1SH2Link:
               /NA/Canada/QC/Montreal/somesite/SH1SH2Link</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u1l1i0</id>
        <identifier>SH1SH2Link Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1h1ieth0</connectedInterface>
        <hostedNetworkInterface>p:c1n1m0l1p1p2i0</hostedNetworkInterface>
        <hostingNetworkInterface>p:p1n1u0l1i0</hostingNetworkInterface>
        <resource>p:p1n1u1l1i0nbw</resource>
        <resource>p:p1n1u1l1i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u1l1i1</id>
        <identifier>SH1SH2Link Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1h2ieth0</connectedInterface>
        <hostedNetworkInterface>p:c1n1m0l1p1p2i1</hostedNetworkInterface>
        <hostingNetworkInterface>p:p1n1u0l1i1</hostingNetworkInterface>
        <resource>p:p1n1u1l1i1nbw</resource>
        <resource>p:p1n1u1l1i1lbw</resource>
```

```
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u1l2</id>
  <identifier>SH1PIP2Link</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u1gl1</graphLabel>
  <hostingNetworkElement>p:p1n1u0l2</hostingNetworkElement>
  <!-- since the Tunnel will be asymmetric (1000 up, 800 down)
    - downlink bw of this link is only partially shared
    - same goes for interfaces -->
  <resource>p:p1n1u1l2ubw</resource>
  <resource>p:p1n1u1l2sdbw</resource>
  <resource>p:p1n1u1l2rdbw</resource>
  <resource>p:p1n1u1l2lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/USA/MA/Boston/somesite/SH1PIP2Link</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
    <Feature>
      <!-- Note: alternative way for annotations of different granularity
        - this could be ,eg, Boston:WashingtonDC:... -->
      <attribute>/position/continent/country/state/city</attribute>
      <value>/NA/USA/MA/Boston</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:p1n1u1l2i0</id>
      <identifier>SH1PIP2Link Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1h1ieth1</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0l2i0</hostingNetworkInterface>
      <resource>p:p1n1u1l2i0nbw</resource>
      <resource>p:p1n1u1l2i0slbw</resource>
      <resource>p:p1n1u1l2i0rlbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:p1n1u1l2i1</id>
      <identifier>SH1PIP2Link Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1p2ieth0</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0l2i1</hostingNetworkInterface>
      <resource>p:p1n1u1l2i1snbw</resource>
      <resource>p:p1n1u1l2i1rnbw</resource>
      <resource>p:p1n1u1l2i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
```

```
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u1l3</id>
  <identifier>SH1SH3Link</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u1gl1</graphLabel>
  <hostingNetworkElement>p:p1n1u0l3</hostingNetworkElement>
  <resource>p:p1n1u1l3ubw</resource>
  <resource>p:p1n1u1l3dbw</resource>
  <resource>p:p1n1u1l3lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/USA/MA/Boston/somesite/SH1SH2Link</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:p1n1u1l3i0</id>
      <identifier>SH1SH3Link Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1h1ieth2</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0l3i0</hostingNetworkInterface>
      <resource>p:p1n1u1l3i0nbw</resource>
      <resource>p:p1n1u1l3i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:p1n1u1l3i1</id>
      <identifier>SH1SH3Link Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1h3ieth0</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0l3i1</hostingNetworkInterface>
      <resource>p:p1n1u1l3i1nbw</resource>
      <resource>p:p1n1u1l3i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u1l4</id>
  <identifier>SH2PIP3Link</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u1gl1</graphLabel>
  <hostingNetworkElement>p:p1n1u0l4</hostingNetworkElement>
  <resource>p:p1n1u1l4ubw</resource>
  <resource>p:p1n1u1l4dbw</resource>
  <resource>p:p1n1u1l4lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/Canada/QC/Montreal/somesite/SH2PIP3Link</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
```

```
      </Feature>
      <Feature>
        <attribute>/position/continent/country/state/city</attribute>
        <value>/NA/Canada/QC/Montreal</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u1l4i0</id>
        <identifier>SH2PIP3Link Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1h2ieth1</connectedInterface>
        <hostingNetworkInterface>p:p1n1u0l4i0</hostingNetworkInterface>
        <resource>p:p1n1u1l4i0nbw</resource>
        <resource>p:p1n1u1l4i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u1l4i1</id>
        <identifier>SH2PIP3Link Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1p3ieth0</connectedInterface>
        <hostingNetworkInterface>p:p1n1u0l4i1</hostingNetworkInterface>
        <resource>p:p1n1u1l4i1nbw</resource>
        <resource>p:p1n1u1l4i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u1l5p1</id>
    <identifier>SH1SH2TunnelPart1</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:p1n1u1gl1</graphLabel>
    <hostedNetworkElement>p:c1n1m0l1p2p2</hostedNetworkElement>
    <resource>p:p1n1u1l2ubw</resource>
    <!-- we share only a part of L2's directional resources -->
    <resource>p:p1n1u1l2sdbw</resource>
    <!-- this could be a properly owned 'resource', if it's different -->
    <resource>p:p1n1u1l2lat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SH1PIP2Link</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u1l5p1i0</id>
        <identifier>SH1SH2Tunnel Part1 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1h1itun0</connectedInterface>
```

```
        <hostedNetworkInterface>p:c1n1m0l1p2p2i0</hostedNetworkInterface>
        <hostingNetworkInterface>p:p1n1u0l2i0</hostingNetworkInterface>
        <resource>p:p1n1u1l2i0nbw</resource>
        <resource>p:p1n1u1l2i0slbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u1l5p1i1</id>
        <identifier>SH1SH2Tunnel Part1 Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1l5p2i0</connectedInterface>
        <hostingNetworkInterface>p:p1n1u0l2i1</hostingNetworkInterface>
        <resource>p:p1n1u1l2i1snbw</resource>
        <resource>p:p1n1u1l2i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <!-- please mind note added to PIP2 and PIP3 w.r.t. L5P2 -->
    <id>p:p1n1u1l5p2</id>
    <identifier>SH1SH2TunnelPart2</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:p1n1u1gl1</graphLabel>
    <hostedNetworkElement>p:c1n1m0l1p2p2</hostedNetworkElement>
    <resource>p:p1n1u1l5p2ubw</resource>
    <resource>p:p1n1u1l5p2dbw</resource>
    <resource>p:p1n1u1l5p2lat</resource>
    <features>
      <Feature>
        <!-- The tunnel resources have been leased with this requirement -->
        <attribute>/position/continent/country</attribute>
        <value>/NA/USA:/NA/Canada</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u1l5p2i0</id>
        <identifier>SH1SH2Tunnel Part2 Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1l5p1i1</connectedInterface>
        <resource>p:p1n1u1l5p2i0nbw</resource>
        <resource>p:p1n1u1l5p2i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u1l5p2i1</id>
        <identifier>SH1SH2Tunnel Part2 Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u1l5p3i0</connectedInterface>
        <resource>p:p1n1u1l5p2i1nbw</resource>
        <resource>p:p1n1u1l5p2i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
```

```
<NetworkElement>
  <id>p:p1n1u1l5p3</id>
  <identifier>SH1SH2TunnelPart3</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u1gl1</graphLabel>
  <hostedNetworkElement>p:c1n1m0l1p2p2</hostedNetworkElement>
  <resource>p:p1n1u1l3ubw</resource>
  <resource>p:p1n1u1l3dbw</resource>
  <resource>p:p1n1u1l3lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/Canada/QC/Montreal/somesite/SH2PIP3Link</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:p1n1u1l5p3i0</id>
      <identifier>SH1SH2Tunnel Part3 Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1l5p2i1</connectedInterface>
      <hostingNetworkInterface>p:p1n1u0l4i1</hostingNetworkInterface>
      <resource>p:p1n1u1l4i1nbw</resource>
      <resource>p:p1n1u1l4i1lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:p1n1u1l5p3i1</id>
      <identifier>SH1SH2Tunnel Part3 Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u1h2itun0</connectedInterface>
      <hostedNetworkInterface>p:c1n1m0l1p2p2i1</hostedNetworkInterface>
      <hostingNetworkInterface>p:p1n1u0l4i0</hostingNetworkInterface>
      <resource>p:p1n1u1l4i0nbw</resource>
      <resource>p:p1n1u1l4i0lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<!-- Resources -->
<Resource>
  <id>p:p1n1u1h1ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h1hd1</id>
  <identifier>Local HDD</identifier>
  <attribute>/host/generic/HDD/space</attribute>
  <value>1000</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
```

```
</Resource>
<Resource>
  <id>p:p1n1u1nfs1</id>
  <identifier>NFS Volume</identifier>
  <attribute>/host/generic/HDD/space</attribute>
  <value>1000</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw
    - we're willing to forward aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u1h1fbw</id>
  <attribute>/node/host/generic/forward/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to handle in virtual splitters
    - aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u1h1sbw</id>
  <attribute>/node/host/generic/splitting/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h1slat</id>
  <attribute>/node/host/splitting/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h1flat</id>
  <attribute>/node/host/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2hd</id>
```

```
  <attribute>/host/generic/HDD/space</attribute>
  <value>1000</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to forward aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u1h2fbw</id>
  <attribute>/node/host/generic/forward/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to handle in virtual splitters
    - aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u1h2sbw</id>
  <attribute>/node/host/generic/splitting/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2slat</id>
  <attribute>/node/host/splitting/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2flat</id>
  <attribute>/node/host/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h3ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to forward aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u1h3fbw</id>
  <attribute>/node/host/generic/forward/bandwidth</attribute>
  <value>20000</value>
```

```
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1h3flat</id>
    <attribute>/node/host/forward/latency</attribute>
    <value>0.5</value>
    <resource_unit>ms</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l1ubw</id>
    <attribute>/link/generic/upstream/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l1dbw</id>
    <attribute>/link/generic/downstream/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l1lat</id>
    <attribute>/link/generic/latency</attribute>
    <value>8</value>
    <resource_unit>ms</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l2ubw</id>
    <attribute>/link/generic/upstream/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l2sdbw</id>
    <attribute>/link/generic/downstream/bandwidth</attribute>
    <value>800</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l2rdbw</id>
    <attribute>/link/generic/downstream/bandwidth</attribute>
    <value>200</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l2lat</id>
```

```
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- for demonstration purposes, this is an asymmetric uplink
    - making bandwidth resource mapping relevant -->
  <id>p:p1n1u1l4ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l4dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l4lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- for demonstration purposes, this is an asymmetric uplink
    - making bandwidth resource mapping relevant -->
  <id>p:p1n1u1l5p2dbw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
```

```
    <value>800</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l5p2ubw</id>
    <attribute>/link/generic/downstream/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l5p2lat</id>
    <attribute>/link/generic/latency</attribute>
    <value>6</value>
    <resource_unit>ms</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1h1ieth0nbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1h1ieth0lbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1h1ieth1snbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>800</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1h1ieth1rnbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>200</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1h1ieth1lbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
```

```
  <id>p:p1n1u1h1ieth2nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h1ieth2lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2ieth1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h2ieth1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h3ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1h3ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
```

```
</Resource>
<Resource>
  <id>p:p1n1u1p2ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1p2ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1p3ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1p3ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l1i1lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
```

```
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l2i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l2i0slbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l2i0rlbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>200</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l2i1slbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l2i1rnbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>200</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3i0lbw</id>
```

```
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3i1lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l3i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l4i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l4i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l4i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l4i1lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u1l5p2i0lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
```

```
  <Resource>
    <id>p:p1n1u1l5p2i0nbw</id>
    <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l5p2i1lbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>1000</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
  <Resource>
    <id>p:p1n1u1l5p2i1nbw</id>
    <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
    <value>800</value>
    <resource_unit>Mbps</resource_unit>
    <request_flag>false</request_flag>
  </Resource>
</Topology>



<Topology>   <!-- UL0 - describing actual substrate -->
  <!-- H3=L3/6=H1-L1-H2 ; H1-L2-P2(...unknown...)P3-L4-H2 -->
  <GraphLabel>
    <!-- modeled NFS resource as shared resource, described tunnel -->
    <id>p:p1n1u0gl1</id>
    <role_identifier>PIP1</role_identifier>
    <graph_type>UL</graph_type>
    <graph_nr>0</graph_nr>
  </GraphLabel>
  <NetworkElement>
    <id>p:p1n1u0h1</id>
    <identifier>SHost1</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:p1n1u0gl1</graphLabel>
    <hostedNetworkElement>p:p1n1u1h1</hostedNetworkElement>
    <resource>p:p1n1u0h1ram</resource>
    <resource>p:p1n1u0h1hd1</resource>
    <resource>p:p1n1u0h1sbw</resource>
    <resource>p:p1n1u0h1slat</resource>
    <resource>p:p1n1u0h1fbw</resource>
    <resource>p:p1n1u0h1flat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SHost1</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
```

```
    <networkInterfaces>
      <!-- direct link to SHost2 in Canada -->
      <NetworkInterface>
        <id>p:p1n1u0h1ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l1i0</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1h1ieth0</hostedNetworkInterface>
        <resource>p:p1n1u0h1ieth0nbw</resource>
        <resource>p:p1n1u0h1ieth0lbw</resource>
      </NetworkInterface>
      <!-- physical link to neighbour Provider PIP2 -->
      <NetworkInterface>
        <id>p:p1n1u0h1ieth1</id>
        <identifier>eth1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l2i0</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1h1ieth1</hostedNetworkInterface>
        <resource>p:p1n1u0h1ieth1nbw</resource>
        <resource>p:p1n1u0h1ieth1lbw</resource>
      </NetworkInterface>
      <!-- physical link to unused SHost3 -->
      <NetworkInterface>
        <id>p:p1n1u0h1ieth2</id>
        <identifier>eth2</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l3i0</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1h1ieth2</hostedNetworkInterface>
        <resource>p:p1n1u0h1ieth2nbw</resource>
        <resource>p:p1n1u0h1ieth2lbw</resource>
      </NetworkInterface>
      <!-- physical link to unused SHost3 via which NFS volume is shared
        - (doesn't appear in UL1) -->
      <NetworkInterface>
        <id>p:p1n1u0h1ieth3</id>
        <identifier>eth3</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l6i0</connectedInterface>
        <resource>p:p1n1u0h1ieth3nbw</resource>
        <resource>p:p1n1u0h1ieth1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u0h2</id>
    <identifier>SHost2</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:p1n1u0gl1</graphLabel>
    <hostedNetworkElement>p:p1n1u1h2</hostedNetworkElement>
    <resource>p:p1n1u0h2ram</resource>
    <resource>p:p1n1u0h2hd</resource>
    <resource>p:p1n1u0h1sbw</resource>
    <resource>p:p1n1u0h1slat</resource>
    <resource>p:p1n1u0h1fbw</resource>
```

```
    <resource>p:p1n1u0h1flat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/Canada/QC/Montreal/somesite/SHost2</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <!-- direct link to SHost 1 -->
      <NetworkInterface>
        <id>p:p1n1u0h2ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l1i1</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1h2ieth0</hostedNetworkInterface>
        <resource>p:p1n1u0l1h2ieth0nbw</resource>
        <resource>p:p1n1u0l1h2ieth0lbw</resource>
      </NetworkInterface>
      <!-- physical link to neighbour Provider PIP3 -->
      <NetworkInterface>
        <id>p:p1n1u0h2ieth1</id>
        <identifier>eth1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l4i0</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1h2ieth1</hostedNetworkInterface>
        <resource>p:p1n1u0l1h2ieth1nbw</resource>
        <resource>p:p1n1u0l1h2ieth1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u0h3</id>
    <identifier>SHost3</identifier>
    <ne_type>/node/host/generic</ne_type>
    <graphLabel>p:p1n1u0gl1</graphLabel>
    <hostedNetworkElement>p:p1n1u1h3</hostedNetworkElement>
    <resource>p:p1n1u0h3ram</resource>
    <resource>p:p1n1u0nfs1</resource>
    <resource>p:p1n1u0h1fbw</resource>
    <resource>p:p1n1u0h1flat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SHost3</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <!-- direct link to SHost 1 -->
      <NetworkInterface>
        <id>p:p1n1u0h3ieth0</id>
```

```
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l3i1</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1h3ieth0</hostedNetworkInterface>
        <resource>p:p1n1u0h3ieth0nbw</resource>
        <resource>p:p1n1u0h3ieth0lbw</resource>
      </NetworkInterface>
      <!-- direct NFS-sharing link to SHost 1 -->
      <NetworkInterface>
        <id>p:p1n1u0h3ieth1</id>
        <identifier>eth1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l6i1</connectedInterface>
        <resource>p:p1n1u0h3ieth1nbw</resource>
        <resource>p:p1n1u0h3ieth1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <!-- This is a neighbouring PIP2 we have an uplink to
      - it's interconneted with PIP3,
      - via which we have an alternative tunnel link to SHost2 -->
    <id>p:p1n1u0p2</id>
    <identifier>PIP2</identifier>
    <ne_type>/node/pip/generic</ne_type>
    <graphLabel>p:p1n1u0gl1</graphLabel>
    <hostedNetworkElement>p:p1n1u1p2</hostedNetworkElement>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city</attribute>
        <value>/NA/USA/MA/Boston</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <!-- direct link to SHost 1 -->
      <NetworkInterface>
        <id>p:p1n1u0p2ieth0</id>
        <identifier>eth0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0l2i1</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1p2ieth0</hostedNetworkInterface>
        <resource>p:p1n1u0p2ieth0nbw</resource>
        <resource>p:p1n1u0p2ieth0lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <!-- This is a neighbouring PIP3 we have an uplink to
      - it's interconneted with PIP2,
      - via which we have an alternative tunnel link to SHost1 -->
    <id>p:p1n1u0p3</id>
    <identifier>PIP3</identifier>
```

```
  <ne_type>/node/pip/generic</ne_type>
  <graphLabel>p:p1n1u0gl1</graphLabel>
  <hostedNetworkElement>p:p1n1u1p3</hostedNetworkElement>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city</attribute>
      <value>/NA/Canada/QC/Montreal</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <!-- direct link to SHost 2 -->
    <NetworkInterface>
      <id>p:p1n1u0p3ieth0</id>
      <identifier>eth0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u0l4i1</connectedInterface>
      <hostedNetworkInterface>p:p1n1u1p3ieth0</hostedNetworkInterface>
      <resource>p:p1n1u0p3ieth0nbw</resource>
      <resource>p:p1n1u0p3ieth0lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u0l1</id>
  <identifier>SH1SH2Link</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u0gl1</graphLabel>
  <hostedNetworkElement>p:p1n1u1l1</hostedNetworkElement>
  <resource>p:p1n1u0l1ubw</resource>
  <resource>p:p1n1u0l1dbw</resource>
  <resource>p:p1n1u0l1lat</resource>
  <features>
    <Feature>
      <!-- Note: in principle, one might list all positions in here
        - to verify position compliance
        - omitted for sake of simplicity -->
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/USA/MA/Boston/somesite/SH1SH2Link:
             /NA/Canada/QC/Montreal/somesite/SH1SH2Link</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:p1n1u0l1i0</id>
      <identifier>SH1SH2Link Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u0h1ieth0</connectedInterface>
      <hostedNetworkInterface>p:p1n1u1l1i0</hostedNetworkInterface>
      <resource>p:p1n1u0l1i0nbw</resource>
      <resource>p:p1n1u0l1i0lbw</resource>
```

```
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u0l1i1</id>
        <identifier>SH1SH2Link Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0h2ieth0</connectedInterface>
        <hostedNetworkInterface>p:p1n110l1i1</hostedNetworkInterface>
        <resource>p:p1n1u0l1i1nbw</resource>
        <resource>p:p1n1u0l1i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u0l2</id>
    <identifier>SH1PIP2Link</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:p1n1u0gl1</graphLabel>
    <hostedNetworkElement>p:p1n1u1l2</hostedNetworkElement>
    <!-- since the Tunnel will be asymmetric (1000 up, 800 down)
      - downlink bw of this link is only partially shared
      - same goes for interfaces -->
    <resource>p:p1n1u0l2ubw</resource>
    <resource>p:p1n1u0l2dbw</resource>
    <resource>p:p1n1u0l2lat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SH1PIP2Link</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
      <Feature>
        <!-- Note: alternative way for annotations of different granularity
          - this could be ,eg, Boston:WashingtonDC:... -->
        <attribute>/position/continent/country/state/city</attribute>
        <value>/NA/USA/MA/Boston</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u0l2i0</id>
        <identifier>SH1PIP2Link Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0h1ieth1</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1l2i0</hostedNetworkInterface>
        <resource>p:p1n1u0l2i0nbw</resource>
        <resource>p:p1n1u0l2i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u0l2i1</id>
        <identifier>SH1PIP2Link Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
```

```
      <connectedInterface>p:p1n1u0p2ieth0</connectedInterface>
      <hostedNetworkInterface>p:p1n1u1l2i1</hostedNetworkInterface>
      <resource>p:p1n1u0l2i1nbw</resource>
      <resource>p:p1n1u0l2i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u0l3</id>
  <identifier>SH1SH3Link</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u0gl1</graphLabel>
  <hostedNetworkElement>p:p1n1u1l3</hostedNetworkElement>
  <resource>p:p1n1u0l3ubw</resource>
  <resource>p:p1n1u0l3dbw</resource>
  <resource>p:p1n1u0l3lat</resource>
  <features>
    <Feature>
      <attribute>/position/continent/country/state/city/site/device</attribute>
      <value>/NA/USA/MA/Boston/somesite/SH1SH2Link</value>
      <priority>1</priority>
      <request_flag>false</request_flag>
    </Feature>
  </features>
  <networkInterfaces>
    <NetworkInterface>
      <id>p:p1n1u0l3i0</id>
      <identifier>SH1SH3Link Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u0h1ieth2</connectedInterface>
      <hostedNetworkInterface>p:p1n1u1l3i0</hostedNetworkInterface>
      <resource>p:p1n1u0l3i0nbw</resource>
      <resource>p:p1n1u0l3i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:p1n1u0l3i1</id>
      <identifier>SH1SH3Link Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u0h3ieth0</connectedInterface>
      <hostedNetworkInterface>p:p1n1u1l3i1</hostedNetworkInterface>
      <resource>p:p1n1u0l3i1nbw</resource>
      <resource>p:p1n1u0l3i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<NetworkElement>
  <id>p:p1n1u0l4</id>
  <identifier>SH2PIP3Link</identifier>
  <ne_type>/link/generic</ne_type>
  <graphLabel>p:p1n1u0gl1</graphLabel>
  <hostingNetworkElement>p:p1n1u1l4</hostingNetworkElement>
  <resource>p:p1n1u0l4ubw</resource>
  <resource>p:p1n1u0l4dbw</resource>
  <resource>p:p1n1u0l4lat</resource>
```

```
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/Canada/QC/Montreal/somesite/SH2PIP3Link</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
      <Feature>
        <attribute>/position/continent/country/state/city</attribute>
        <value>/NA/Canada/QC/Montreal</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u0l4i0</id>
        <identifier>SH2PIP3Link Interface 0</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0h2ieth1</connectedInterface>
        <hostedNetworkInterface>p:p1n1u1l4i0</hostedNetworkInterface>
        <resource>p:p1n1u0l4i0nbw</resource>
        <resource>p:p1n1u0l4i0lbw</resource>
      </NetworkInterface>
      <NetworkInterface>
        <id>p:p1n1u0l4i1</id>
        <identifier>SH2PIP3Link Interface 1</identifier>
        <ni_type>/interface/generic</ni_type>
        <connectedInterface>p:p1n1u0p3ieth0</connectedInterface>
        <hostingNetworkInterface>p:p1n1u1l4i1</hostingNetworkInterface>
        <resource>p:p1n1u0l4i1nbw</resource>
        <resource>p:p1n1u0l4i1lbw</resource>
      </NetworkInterface>
    </networkInterfaces>
  </NetworkElement>
  <NetworkElement>
    <id>p:p1n1u0l6</id>
    <identifier>SH1SH3NFSLink</identifier>
    <ne_type>/link/generic</ne_type>
    <graphLabel>p:p1n1u0gl1</graphLabel>
    <resource>p:p1n1u0l6ubw</resource>
    <resource>p:p1n1u0l6dbw</resource>
    <resource>p:p1n1u0l6lat</resource>
    <features>
      <Feature>
        <attribute>/position/continent/country/state/city/site/device</attribute>
        <value>/NA/USA/MA/Boston/somesite/SH1SH2NFSLink</value>
        <priority>1</priority>
        <request_flag>false</request_flag>
      </Feature>
    </features>
    <networkInterfaces>
      <NetworkInterface>
        <id>p:p1n1u0l6i0</id>
```

```
      <identifier>SH1SH3NFSLink Interface 0</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u0h1ieth3</connectedInterface>
      <resource>p:p1n1u0l6i0nbw</resource>
      <resource>p:p1n1u0l6i0lbw</resource>
    </NetworkInterface>
    <NetworkInterface>
      <id>p:p1n1u0l6i1</id>
      <identifier>SH1SH3NFSLink Interface 1</identifier>
      <ni_type>/interface/generic</ni_type>
      <connectedInterface>p:p1n1u0h3ieth1</connectedInterface>
      <resource>p:p1n1u0l6i1nbw</resource>
      <resource>p:p1n1u0l6i1lbw</resource>
    </NetworkInterface>
  </networkInterfaces>
</NetworkElement>
<!-- Resources -->
<Resource>
  <id>p:p1n1u0h1ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1hd1</id>
  <identifier>Local HDD</identifier>
  <attribute>/host/generic/HDD/space</attribute>
  <value>1000</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0nfs1</id>
  <identifier>NFS Volume</identifier>
  <attribute>/host/generic/HDD/space</attribute>
  <value>1000</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to forward aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u0h1fbw</id>
  <attribute>/node/host/generic/forward/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to handle in virtual splitters
    - aside from hosting nodes
    - could as well model CPU resources,
```

```
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u0h1sbw</id>
  <attribute>/node/host/generic/splitting/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1slat</id>
  <attribute>/node/host/splitting/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1flat</id>
  <attribute>/node/host/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2hd</id>
  <attribute>/host/generic/HDD/space</attribute>
  <value>1000</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to forward aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u0h2fbw</id>
  <attribute>/node/host/generic/forward/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to handle in virtual splitters
    - aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u0h2sbw</id>
  <attribute>/node/host/generic/splitting/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
```

```
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2slat</id>
  <attribute>/node/host/splitting/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2flat</id>
  <attribute>/node/host/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h3ram</id>
  <attribute>/host/generic/RAM/real/amount</attribute>
  <value>10</value>
  <resource_unit>GB</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- amount of bw we're willing to forward
    - aside from hosting nodes
    - could as well model CPU resources,
    - if we knew sensible units and costs associated to bw hosting -->
  <id>p:p1n1u0h3fbw</id>
  <attribute>/node/host/generic/forward/bandwidth</attribute>
  <value>20000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h3flat</id>
  <attribute>/node/host/forward/latency</attribute>
  <value>0.5</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l1ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l1dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
```

```
</Resource>
<Resource>
  <id>p:p1n1u0l1lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>8</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l2ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l2dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l2lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <!-- for demonstration purposes, this is an asymmetric uplink
    - making bandwidth resource mapping relevant -->
  <id>p:p1n1u0l4ubw</id>
```

```
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l4dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l4lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l6ubw</id>
  <attribute>/link/generic/upstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l6dbw</id>
  <attribute>/link/generic/downstream/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l6lat</id>
  <attribute>/link/generic/latency</attribute>
  <value>1</value>
  <resource_unit>ms</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
```

```
<Resource>
  <id>p:p1n1u0h1ieth1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth2nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth2lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth3nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h1ieth3lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
```

```
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2ieth1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h2ieth1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h3ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h3ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h3ieth1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0h3ieth1lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0p2ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0p2ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
```

```
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0p3ieth0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0p3ieth0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l1i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l1i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l1i1lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l1i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l2i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
```

```
  <id>p:p1n1u0l2i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l2i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l2i1lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3i0lbw</id>
  <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3i1lbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l3i1nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>1000</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
</Resource>
<Resource>
  <id>p:p1n1u0l4i0nbw</id>
  <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
  <value>800</value>
  <resource_unit>Mbps</resource_unit>
  <request_flag>false</request_flag>
```

```
    </Resource>
    <Resource>
      <id>p:p1n1u0l4i0lbw</id>
      <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
      <value>1000</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
    <Resource>
      <id>p:p1n1u0l4i1nbw</id>
      <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
      <value>1000</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
    <Resource>
      <id>p:p1n1u0l4i1lbw</id>
      <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
      <value>800</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
    <Resource>
      <id>p:p1n1u0l6i0nbw</id>
      <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
      <value>1000</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
    <Resource>
      <id>p:p1n1u0l6i0lbw</id>
      <attribute>/interface/generic/ne_to_connection/bandwidth</attribute>
      <value>1000</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
    <Resource>
      <id>p:p1n1u0l6i1lbw</id>
      <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
      <value>1000</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
    <Resource>
      <id>p:p1n1u0l6i1nbw</id>
      <attribute>/interface/generic/connection_to_ne/bandwidth</attribute>
      <value>1000</value>
      <resource_unit>Mbps</resource_unit>
      <request_flag>false</request_flag>
    </Resource>
  </Topology>
```

```
<!-- ResourceMappings -->
<!-- OL0 -> OL1
  - could optionally map p:c1n1o0r1 to all o1 resources
  - not done for the sake of readability -->
<!-- OL1 -> ML0 -->
<ResourceMapping>
  <hostedResource>p:c1n1o1h1ram</hostedResource>
  <hostingResource>p:c1n1m0h1ram</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h1hd</hostedResource>
  <hostingResource>p:c1n1m0h1hd</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h2ram</hostedResource>
  <hostingResource>p:c1n1m0h2ram</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h2hd</hostedResource>
  <hostingResource>p:c1n1m0h2hd</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1bw</hostedResource>
  <hostingResource>p:c1n1m0l1s1bw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1lat</hostedResource>
  <hostingResource>p:c1n1m0l1s1lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1ubw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p1ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1dbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p1dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>0.5</value>
  <hostedResource>p:c1n1o1l1p1lat</hostedResource>
  <hostingResource>p:c1n1m0l1p1p1lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1ubw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p2ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1dbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p2dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>8</value>
  <hostedResource>p:c1n1o1l1p1lat</hostedResource>
  <hostingResource>p:c1n1m0l1p1p2lat</hostingResource>
```

```
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1ubw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p3ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1dbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p3dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>0.5</value>
  <hostedResource>p:c1n1o1l1p1lat</hostedResource>
  <hostingResource>p:c1n1m0l1p1p3lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2ubw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p1ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2dbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p1dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>0.5</value>
  <hostedResource>p:c1n1o1l1p2lat</hostedResource>
  <hostingResource>p:c1n1m0l1p2p1lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2ubw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p2ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2dbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p2dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>8</value>
  <hostedResource>p:c1n1o1l1p2lat</hostedResource>
  <hostingResource>p:c1n1m0l1p2p2lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2ubw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p3ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2dbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p3dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>0.5</value>
  <hostedResource>p:c1n1o1l1p2lat</hostedResource>
  <hostingResource>p:c1n1m0l1p2p3lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
```

```
  <hostedResource>p:c1n1o1l1s2bw</hostedResource>
  <hostingResource>p:c1n1m0l1s2bw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2lat</hostedResource>
  <hostingResource>p:c1n1m0l1s2lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h1ieth0nbw</hostedResource>
  <hostingResource>p:c1n1m0h1ieth0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h1ieth0lbw</hostedResource>
  <hostingResource>p:c1n1m0h1ieth0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1i0nbw</hostedResource>
  <hostingResource>p:c1n1m0l1s1i0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1i0lbw</hostedResource>
  <hostingResource>p:c1n1m0l1s1i0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1i1nbw</hostedResource>
  <hostingResource>p:c1n1m0l1s1i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1i1lbw</hostedResource>
  <hostingResource>p:c1n1m0l1s1i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1i2nbw</hostedResource>
  <hostingResource>p:c1n1m0l1s1i2nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s1i2lbw</hostedResource>
  <hostingResource>p:c1n1m0l1s1i2lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1i0nbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p1i0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1i0lbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p1i0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1i1nbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p3i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p1i1lbw</hostedResource>
  <hostingResource>p:c1n1m0l1p1p3i1lbw</hostingResource>
```

```
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2i0nbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p1i0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2i0lbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p1i0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2i1nbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p3i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1p2i1lbw</hostedResource>
  <hostingResource>p:c1n1m0l1p2p3i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2i0nbw</hostedResource>
  <hostingResource>p:c1n1m0l1s2i0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2i0lbw</hostedResource>
  <hostingResource>p:c1n1m0l1s2i0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2i1nbw</hostedResource>
  <hostingResource>p:c1n1m0l1s2i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2i1lbw</hostedResource>
  <hostingResource>p:c1n1m0l1s2i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2i2nbw</hostedResource>
  <hostingResource>p:c1n1m0l1s2i2nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1l1s2i2lbw</hostedResource>
  <hostingResource>p:c1n1m0l1s2i2lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h2ieth0nbw</hostedResource>
  <hostingResource>p:c1n1m0h2ieth0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1o1h2ieth0lbw</hostedResource>
  <hostingResource>p:c1n1m0h2ieth0lbw</hostingResource>
</ResourceMapping>
<!-- ML0 -> UL1 -->
<ResourceMapping>
  <hostedResource>p:c1n1m0h1ram</hostedResource>
  <hostingResource>p:p1n1u1h1ram</hostingResource>
</ResourceMapping>
```

```
<ResourceMapping>
  <!-- since SHost1 actually has two HDD resources,
    - we actually need the Resource mapping here
    - to identify the correct one -->
  <hostedResource>p:c1n1m0h1hd</hostedResource>
  <hostingResource>p:p1n1u1nfs1</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0h2ram</hostedResource>
  <hostingResource>p:p1n1u1h2ram</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0h2hd</hostedResource>
  <hostingResource>p:p1n1u1h2hd</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1bw</hostedResource>
  <hostingResource>p:p1n1u1h1sbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1lat</hostedResource>
  <hostingResource>p:p1n1u1h1slat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p1ubw</hostedResource>
  <hostingResource>p:p1n1u1h1fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p1dbw</hostedResource>
  <hostingResource>p:p1n1u1h1fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p1lat</hostedResource>
  <hostingResource>p:p1n1u1h1flat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2ubw</hostedResource>
  <hostingResource>p:p1n1u1l1ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2dbw</hostedResource>
  <hostingResource>p:p1n1u1l1dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2lat</hostedResource>
  <hostingResource>p:p1n1u1l1lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p3ubw</hostedResource>
  <hostingResource>p:p1n1u1h2fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p3dbw</hostedResource>
  <hostingResource>p:p1n1u1h2fbw</hostingResource>
```

```
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p3lat</hostedResource>
  <hostingResource>p:p1n1u1h2flat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p1ubw</hostedResource>
  <hostingResource>p:p1n1u1h1fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p1dbw</hostedResource>
  <hostingResource>p:p1n1u1h1fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p1lat</hostedResource>
  <hostingResource>p:p1n1u1h1flat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2ubw</hostedResource>
  <hostingResource>p:p1n1u1l5p1ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2dbw</hostedResource>
  <hostingResource>p:p1n1u1l5p1dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>1</value>
  <hostedResource>p:c1n1m0l1p2p2lat</hostedResource>
  <hostingResource>p:p1n1u1l5p1lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2ubw</hostedResource>
  <hostingResource>p:p1n1u1l5p2ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2dbw</hostedResource>
  <hostingResource>p:p1n1u1l5p2dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>6</value>
  <hostedResource>p:c1n1m0l1p2p2lat</hostedResource>
  <hostingResource>p:p1n1u1l5p2lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2ubw</hostedResource>
  <hostingResource>p:p1n1u1l5p3ubw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2dbw</hostedResource>
  <hostingResource>p:p1n1u1l5p3dbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <value>1</value>
  <hostedResource>p:c1n1m0l1p2p2lat</hostedResource>
```

```
  <hostingResource>p:p1n1u1l5p3lat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p3ubw</hostedResource>
  <hostingResource>p:p1n1u1h2fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p3dbw</hostedResource>
  <hostingResource>p:p1n1u1h2fbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p3lat</hostedResource>
  <hostingResource>p:p1n1u1h2flat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s2bw</hostedResource>
  <hostingResource>p:p1n1u1h1sbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s2lat</hostedResource>
  <hostingResource>p:p1n1u1h1slat</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1i0nbw</hostedResource>
  <hostingResource>p:p1n1u1l1s1i0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1i0lbw</hostedResource>
  <hostingResource>p:p1n1u1l1s1i0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1i1nbw</hostedResource>
  <hostingResource>p:p1n1u1l1s1i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1i1lbw</hostedResource>
  <hostingResource>p:p1n1u1l1s1i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1i2nbw</hostedResource>
  <hostingResource>p:p1n1u1l1s1i2nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1s1i2lbw</hostedResource>
  <hostingResource>p:p1n1u1l1s1i2lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p1i1nbw</hostedResource>
  <hostingResource>p:p1n1u1h1ieth0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p1i1lbw</hostedResource>
  <hostingResource>p:p1n1u1h1ieth0lbw</hostingResource>
</ResourceMapping>
```

```
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2i0nbw</hostedResource>
  <hostingResource>p:p1n1u1l1i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2i0lbw</hostedResource>
  <hostingResource>p:p1n1u1l1i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2i1nbw</hostedResource>
  <hostingResource>p:p1n1u1l1i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p2i1lbw</hostedResource>
  <hostingResource>p:p1n1u1l1i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p3i0nbw</hostedResource>
  <hostingResource>p:p1n1u1h2ieth0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p1p3i0lbw</hostedResource>
  <hostingResource>p:p1n1u1h2ieth0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p1i1nbw</hostedResource>
  <hostingResource>p:p1n1u1h1ieth1snbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p1i1lbw</hostedResource>
  <hostingResource>p:p1n1u1h1ieth1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2i0nbw</hostedResource>
  <hostingResource>p:p1n1u1l5p1i0nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2i0lbw</hostedResource>
  <hostingResource>p:p1n1u1l5p1i0lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2i1nbw</hostedResource>
  <hostingResource>p:p1n1u1l5p3i1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p2i1lbw</hostedResource>
  <hostingResource>p:p1n1u1l5p3i1lbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p3i0nbw</hostedResource>
  <hostingResource>p:p1n1u1h2ieth1nbw</hostingResource>
</ResourceMapping>
<ResourceMapping>
  <hostedResource>p:c1n1m0l1p2p3i0lbw</hostedResource>
```

```
    <hostingResource>p:p1n1u1h2ieth1lbw</hostingResource>
  </ResourceMapping>
  <!-- UL1 -> UL0 -->
  <!-- canonic - omitted for sake of brevity ->
</root>
```

Acknowledgements

First and foremost, I would like to thank my referees for accepting the time-consuming task of reviewing my thesis. Likewise, I would like to thank Alexander Raake, who presided over my defense.

I am extremely grateful to my advisor Anja Feldmann. Since her first lecture in Saarbrücken, where she sparked my interest in computer networking, I've come to respect and like her as advisor, boss, and as a person. I'd like to thank her for all the splendid, direct discussions - which where sometimes tough, but invariably meant a step forward in what mattered. Her motivation, guidance, and the fact that one could rely on her in every way, created a work environment I sincerely regret leaving, but which I will always keep in good memory. Besides, her professional migrations provided me with good reasons to live and work in different, interesting corners of Germany.

Next, I want to thank my various colleagues and collaborators at FG INET, Docomo EuroLabs, and elsewhere. Special thanks go to my friends and collaborators Carlo, Stefan, Johannes and Ernesto. Their help made it possible to cope with project workload and to work on a broader field than otherwise possible. Moreover, the discussions both inside FG INET and outside always made me reconsider purpose and context of my work, and thereby helped in increasing its consistency.

I'd also like to express my gratitude to friends like Philip Schneider, Tim Dahmen, Harald and Doris Schiöberg. Discussing issues - private and professional... listing to their external view on my situation, often helped me to understand and learn from my mistakes.

Last, but not least, there were three essential factors that enabled me to finish my thesis: The love, humour, and wits of my partner Stella, the caffeine in Mate, provided by my friend and former colleague Fabio Hecht, and the continuous support of my mother, without whom I would not stand where I am today.

In the end, a word to FG INET: So long, and thanks for all the hat! I've enjoyed working with you, and hope to stay in contact!

List of Figures

We use abbrevations for the most common plots in this theses: *(i)* Probability density functions (PDFs), *(ii)* Cumulative distribution functions (CDFs), and *(iii)* Complementary cumulative distribution functions (CCDFs).

List of Tables

Bibliography

[1] VINI End User Attachment. http://www.vini-veritas.net/documentation/pl-vini/user/clients.

[2] 802.1q vlan, June 2009. http://standards.ieee.org/getieee802/.

[3] Anual Report DTAG 2008. http://www.download-telekom.de/dt/StaticPage/62/37/50/090227_DTAG_2008_Annual_report.pdf_623750.pdf, 2009.

[4] AT&T Report on planned investments Mar, 10 2009. http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=26597, 2009.

[5] AT&T Reports Fourth-Quarter and Full-Year Results 2008. http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=26502, 2009.

[6] Cloud computing: An overview. *Queue 7*, 5 (June 2009), 2:3–2:4.

[7] MySQL project website, Dec. 2009. http://www.mysql.de.

[8] Netcat Website, Dec. 2009. http://netcat.sourceforge.net.

[9] XMLRPC Specifications Website, Dec. 2009. http://www.xmlrpc.com/spec.

[10] YAML Website, Dec. 2009. http://www.yaml.org/.

[11] Amazon virtual private cloud adds ip address assignment capability. http://aws.amazon.com/about-aws/whats-new/2010/07/13/amazon-virtual-private-cloud-adds-ip-address-assignment/, 2010.

[12] The 2011 financial year. whenever. whatever. wherever. http://www.e-paper.telekom.com/reports/2011/epaper-GB_2011_en/page126.html, 2012.

[13] Activerecord reference, Apr. 2012. http://api.rubyonrails.org/classes/ActiveRecord/Base.html.

[14] lighttpd website, Apr. 2012. http://www.lighttpd.net.

[15] Netfilter project website, 2012. http://www.netfilter.org.

[16] Openvswitch website, Apr. 2012. http://openvswitch.org.

[17] Ruby website, Apr. 2012. http://www.ruby-lang.org/en/.

[18] Videolan client (vlc) website, Apr. 2012. http://www.videolan.org/vlc/.

[19] A. Kivity, Y. Kamay, D. L., Lublin, U., and Ligouri, A. kvm: The linux virtual machine monitor. In *Proceedings of the Linux Symposium* (2007).

[20] Andersen, D. Theoretical approaches to node assignment. In *CS CMU Report* (2002).

[21] Andersen, E., and Andersen, K. D. Presolving in linear programming. *Mathematical Programming 71*, 2 (1995), 221–245.

[22] Anderson, D. Theoretical approaches to node assignment. Computer Science Department. Paper 86 ; `http://repository.cmu.edu/compsci/86`, 2002.

[23] Anderson, N. Your new isp? google launches 1gbps fiber-to-the-home trial. `http://arstechnica.com/tech-policy/news/2010/02/your-new-isp-google-launches-1gbps-fiber-to-the-home-trial.ars`.

[24] Anderson, T., Peterson, L., Shenker, S., and Turner, J. Overcoming the internet impasse through virtualization. *Computer 38*, 4 (2004).

[25] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM 53*, 4 (Apr. 2010), 50–58.

[26] Arora, D., Bienkowski, M., Feldmann, A., Schaffrath, G., and Schmid, S. Online strategies for intra and inter provider service migration in virtual networks. In *Proceedings on Principles, Systems and Applications of IP Telecommunications (IPTComm) conference* (Chicago, USA, August 2011).

[27] Arora, D., Bienkowski, M., Feldmann, A., Schaffrath, G., and Schmid, S. Online strategies for intra and inter provider service migration in virtual networks. Technical report, arXiv, March 2011. ID arXiv:1103.0966v1 [cs.NI].

[28] Arora, D., Feldmann, A., Schaffrath, G., and Schmid, S. On the benefit of virtualization: Strategies for flexible server allocation. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)* (Boston, USA, March 2011).

[29] Baldine, I., Yufeng, X., Mandal, A., Renci, C., Chase, U., Marupadi, V., Yumerefendi, A., and Irwin, D. Networked cloud orchestration: A geni perspective. In *GC Workshops, 2010 IEEE, Miami, USA* (Dec 2010).

[30] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. XEN and the art of virtualization. In *SOSP'03 Proceedings of the nineteenth ACM symposium on Operating systems principles* (2003), ACM.

[31] Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. Operating system support for planetary-scale network services. In *NSDI Usenix* (2004), ACM.

[32] Bavier, A., Feamster, N., Huang, M., Peterson, L., and Rexford, J. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proceedings of SIGCOMM '06* (2006), ACM, pp. 3–14.

[33] Bhatia, S., Motiwala, M., Mühlbauer, W., Mundada, Y., Valancius, V., Bavier, A., Feamster, N., Peterson, L., and Rexford, J. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *CoNEXT '08 Proceedings of the 2008 ACM CoNEXT Conference* (2008), ACM.

[34] Bienkowski, M. Migrating and replicating data in networks. *Computer Science - Research and Development* (2011).

[35] Bienkowski, M., Feldmann, A., Jurca, D., Kellerer, W., Schaffrath, G., Schmid, S., and Widmer, J. Competitive analysis for service migration in vnets. In *Proc. ACM SIGCOMM VISA Workshop* (New Delhi, India, August 2010), ACM, pp. 17–24.

[36] Bienkowski, M., and Schmid, S. Online function tracking with generalized penalties. In *Proc. 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)* (2010).

[37] Bless, R., Röhricht, M., and Werle, C. Authenticated setup of virtual links with quality-of-service guarantees. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on* (2011).

[38] Borodin, A., and El-Yaniv, R. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[39] Borodin, A., Linial, N., and Saks, M. E. An optimal on-line algorithm for metrical task system. *J. ACM 39*, 4 (1992), 745–763.

[40] Bradford, R., Kotsovinos, E., Feldmann, A., and Schioeberg, H. Live wide-area migration of virtual machines including local persistent state. In *Proc. VEE* (2007).

[41] Breitgand, D., Kutiel, G., and Raz, D. Cost-aware live migration of services in the cloud. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)* (2011).

[42] Butt, N. F., Chowdhury, M., and Boutaba, R. Topology-awareness and reoptimization mechanism for virtual network embedding. In *Proc. IFIP/TC6 NETWORKING* (2010).

[43] Campbell, A., Meer, H. D., Kounavis, M. E., Miki, K., Vincente, J., and Villela, D. A survey of programmable networks. *ACM SIGCOMM Computer Communication Review 29*, 2 (1999).

[44] Chowdhury, K., Rahman, M. R., and Boutaba, R. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM* (2009).

[45] Chowdhury, M., Samuel, F., and Boutaba, R. PolyViNE: Policy-based virtual network embedding across multiple domains. In *Proc. ACM SIGCOMM VISA* (2010).

[46] Chowdhury, M. K., and Boutaba, R. A survey of network virtualization. *Elsevier Computer Networks 54*, 5 (2010).

[47] cker Chiueh, T., and Nanda, S. A survey on virtualization technologies. http://class.ece.iastate.edu/tyagi/cpre681/papers/VirtualizationSurveyTR179.pdf, 2005.

[48] Clark, C., Fraser, K., H, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live Migration of Virtual Machines. In *Proceedings of ACM/USENIX NSDI '05* (2005), pp. 273–286.

[49] Clark, D. Network neutrality: Words of power and 800-pound gorillas. *International Journal of Communication 1* (2007).

[50] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. SIGCOMM* (2002).

[51] Clarke, G. Apple's icloud runs on microsoft and amazon services. http://www.theregister.co.uk/2011/09/02/icloud_runs_on_microsoft_azure_and_amazon/, 2011.

[52] Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. An improved algorithm for matching large graphs. In *Proc. Workshop on Graph-based Representations in Pattern Recognition* (2001).

[53] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms*. MIT press, 2001.

[54] Dike, J. A usermode port for the linux kernel. In *LINUX Showcase and Conference* (2000), ACM.

[55] D.J. Wetherall, J.V. Guttag, D. T. Ants: a toolkit for building and dynamically deploying networkprotocols. In *Open Architectures and Network Programming, 1998 IEEE* (1998), IEEE, pp. 117–129.

[56] et al, A. C. The genesis kernel: a virtual network operating system for spawningnetwork architectures. In *Open Architectures and Network Programming Proceedings, 1999. OPENARCH apos;99. 1999 IEEE Second Conference on* (1999), pp. 115–127.

[57] Even, G., Medina, M., Schaffrath, G., and Schmid, S. Competitive and deterministic embeddings of virtual networks. In *ArXiv Technical Report 1101.5221* (2011).

[58] Even, G., Medina, M., Schaffrath, G., and Schmid, S. Competitive and deterministic embeddings of virtual networks. *Distributed Computing and Networking Lecture Notes in Computer Science 7129* (2012), pp 106–121.

[59] Even, G., Medina, M., Schaffrath, G., and Schmid, S. Competitive and deterministic embeddings of virtual networks. In *Proceedings of the 13th International Conference on Distributed Computing and Networking (ICDCN)* (2012), IEEE.

[60] Fajjari, I., Ayari, M., and Pujolle, G. Vn-sla: A virtual network specification schema for virtual network provisioning. In *Networks (ICN), 2010 Ninth International Conference on* (2010).

[61] Fajjari, I., Ayari, M., Pujolle, G., and Zimmermann, H. Towards an autonomic piloting virtual network architecture. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on* (2011), IEEE.

[62] Fan, J., and Ammar, M. H. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *Proc. IEEE INFOCOM* (2006).

[63] Feamster, N., Gao, L., and Rexford, J. How to Lease the Internet in Your Spare Time. *SIGCOMM CCR 37*, 1 (2007), 61–64.

[64] Feldmann, A. Internet clean-slate design: what and why? *ACM SIGCOMM Computer Communication Review 37*, 3 (2007).

[65] Feldmann, A., Kind, M., Maennel, O., Schaffrath, G., and Werle, C. Network virtualization - an enabler for overcoming ossification. *ERCIM NEWS 77* (April 2009), pp. 21–22.

[66] Feldmann, A., Schaffrath, G., and Schmid, S. Cloudnets: Combining clouds with networking. *ERCIM NEWS 88* (December 2011).

[67] Fotakis, D. On the competitive ratio for online facility location. In *Proc. 30th International Conference on Automata, Languages and Programming (ICALP), also appeared in Algorithmica 50(1), pp. 1-57, 2008* (2003), pp. 637–652.

[68] Fürst, C. Approximations for virtual network embeddings. Diploma thesis, TU Berlin, 2011.

[69] GENI: Global Environment for Network Innovations, 2012. http://www.geni.net.

[70] Guo, C., Lu, G., Wang, H., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *ACM CoNEXT 2010* (2010).

[71] Haider, A., Potter, R., and Nakao, A. Challenges in resource allocation in network virtualization. In *Proc. ITC Specialist Seminar on Network Virtualization* (2009).

[72] Hajjat, M., Sun, X., Sung, Y.-W. E., Maltz, D., Sripanidkulchai, S. R. K., and Tawarmalani, M. Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud. In *Proc. ACM SIGCOMM* (2011).

[73] Hao, F., Lakshman, T. V., Mukherjee, S., and Song, H. Enhancing dynamic cloud-based services using network virtualization. *SIGCOMM Comput. Commun. Rev. 40*, 1 (2010), 67–74.

[74] Harris, D. How the cloud could boost amazon's slow-moving margins, Apr. 2012. `http://gigaom.com/cloud/how-cloud-computing-could-boost-amazons-margins/`.

[75] Holt, A., Weiss, K., Huberty, K., Gelblum, E., Flannery, S., Devgan, S., Malik, A., Rozof, N., Wood, A., Standaert, P., Meunier, F., Lu, J., Chen, G., Lu, B., Han, K., Khare, V., and Miyachi, M. Cloud computing takes off, May 2011.

[76] Houidi, I., Louati, W., and Zeghlache, D. A distributed virtual network mapping algorithm. In *ICC* (2008).

[77] Houidi, I., Louati, W., Zeghlache, D., and Bauke, S. Virtual resource description and clustering for virtual network discovery. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on* (2009).

[78] Jünger, M., Liebling, T., Naddef, D., Nemhauser, G., Pulleyblank, W., Reinelt, G., Rinaldi, G., and (eds.), L. W. *50 Years of Integer Programming 1958-2008*. Springer, 2009.

[79] Kleinberg, J. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems 15* (2002), MIT Press, pp. 446–453.

[80] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., and Wolter, K. Miplib 2010. *Math. Prog. Comp.*, 3 (2011), 103–163.

[81] Koch, T., Ralphs, T., and Shinano, Y. Could we use a million cores to solve an integer program? `http://coral.ie.lehigh.edu/~ted/files/papers/Million11.pdf`, 2011.

[82] Kulkarni, A. B., Minden, G. J., Hill, R., Wijata, Y., Gopinath, A., Sheth, S., Wahhab, F., Pindi, H., and Nagarajan, A. Implementation of a prototype active network. In *Open Architectures and Network Programming* (1998), IEEE, pp. 130–142.

[83] Kumar, A., Rastogi, R., Silberschatz, A., and Yener, B. Algorithms for provisioning virtual private networks in the hose model. *IEEE/ACM Trans. Netw. 10*, 4 (2002).

[84] Laoutaris, N., Smaragdakis, G., Oikonomou, K., Stavrakakis, I., and Bestavros, A. Distributed placement of service facilities in large-scale networks. In *Proc. IEEE INFOCOM* (2007).

[85] Lischka, J., and Karl, H. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. ACM SIGCOMM VISA* (2009).

[86] Lu, J., and Turner, J. Efficient mapping of virtual networks onto a shared substrate. In *WUCSE-2006-35, Washington University* (2006).

[87] Lymberopoulos, L., Grosso, P., Papagianni, C., Kalogeras, D., Androulidakis, G., van der Ham, J., de Laat, C., and Maglaris, V. Managing federations of virtualized infrastructures: A semantic-aware policy based approach. In *Proc. 2011 IFIP/IEEE ISINM, Dublin, Ireland* (May 2011).

[88] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (2008), 69–74.

[89] Merritt, R. Google describes its openflow network. `http://www.eetimes.com/electronics-news/4371179/Google-describes-its-OpenFlow-network`, 2012.

[90] Metsch, T., and Edmonds, A. Gfd 184: Open cloud computing interface-infrastructure. *Open Grid Forum Standards*.

[91] Metsch, T., and Edmonds, A. Gfd 185: Open cloud computing interface-http rendering. *Open Grid Forum Standards*.

[92] Mittelmann, H. Mixed integer linear programming benchmark (miplib2010). `http://plato.asu.edu/ftp/milpc.html`, 2012.

[93] Oberle, K., Kessler, M., Stein, M., Voith, T., Lamp, D., and Berger, S. Network virtualization: The missing piece. In *Intelligence in Next Generation Networks, 2009. ICIN 2009. 13th International Conference on* (Oct. 2009), IEEE.

[94] Openvpn project web page, 2012. `http://openvpn.net/`.

[95] Park, K.-M., and Kim, C.-K. A framework for virtual network embedding in wireless networks. In *Proc. 4th International Conference on Future Internet Technologies (CFI)* (2009), pp. 5–7.

[96] Patterson, D., Clark, D., Karlin, A., Kurose, J., Lazowska, E., Liddle, D., McAuley, D., Paxson, V., Savage, S., and Zegura, E. In *Looking Over the Fence at Networks: A Neighbor's View of Networking Research* (2001), National Academy Press, Washington D.C.

[97] Perkins, D. Understanding SNMP MIBs. In *SynOptics Communication* (1993).

[98] Popek, G. J., and Goldberg, R. P. Formal requirements for virtualizable third generation architectures. *Communications of the ACM 17*, 7 (1974).

[99] Potter, R., and Nakao, A. Mobitopolo: A portable infrastructure to facilitate flexible deployment and migration of distributed applications with virtual topologies. In *Proc. ACM SIGCOMM VISA* (2009), pp. 19–28.

[100] Psounis, K. Active networks: Applications, security, safety, and architectures. *Communications Surveys & Tutorials 2*, 1 (1999).

[101] Remote authentication dial in user service (radius), 2012. `http://www.rfc-editor.org/rfc/rfc2865.txt`.

[102] Ralf, N., Edmonds, A., Papaspyrou, A., and Metsch, T. Gfd 183: Open cloud computing interface-core. *Open Grid Forum Standards*.

[103] Resource description language, 2004. http://www.rfc-editor.org/rfc/rfc3870.txt.

[104] Ricci, R., Alfeld, C., and Lepreau, J. A solver for the network testbed mapping problem. *ACM SIGCOMM CCR 33*, 2 (2003).

[105] Rochwerger, B., Breitgand, D., Epstein, A., Hadas, D., Loy, I., Nagin, K., Tordsson, J., Ragusa, C., Villari, M., Clayman, S., Levy, E., Maraschini, A., Massonet, P., Munoz, H., and Toffetti, G. Reservoir wwhen one cloud is not enough. *IEEE Computer* (2011).

[106] Rose, R. Survey of system virtualization techniques. `http://hdl.handle.net/1957/9907`, 2004.

[107] Rspec v.2, 2012. http://www.protogeni.net/trac/protogeni/wiki/RSpec.

[108] S. da Silva, Y. Yemini, D. F. The netscript active network system. *Selected Areas in Communications 19*, 3 (Mar 2001), 538–551.

[109] Schaffrath, G., Schmid, S., and Feldmann, A. Generalized and resource-efficient vnet embeddings with migrations. *CoRR abs/1012.4066* (2010).

[110] Schaffrath, G., Schmid, S., and Feldmann, A. Optimizing long-lived cloudnets with migrations. In *Proceedings International Conference on Utility and Cloud Computing (UCC)* (Chicago, USA, November 2012).

[111] Schaffrath, G., Schmid, S., Grassler, J., Abarca, E., and Feldmann, A. A virtual network architecture prototype. Tech. rep., to appear.

[112] Schaffrath, G., Schmid, S., Vaishnavi, I., Khan, A., and Feldmann, A. A resource description language with vagueness support for multi-provider cloud networks. In *International Conference on Computer Communication Networks (ICCCN)* (July 2012), pp. pp. 1–7.

[113] Schaffrath, G., Werle, C., Papadimitriou, P., Feldmann, A., Bless, R., Greenhalgh, A., Wundsam, A., Kind, M., Maennel, O., and Mathy, L. Network virtualization architecture: Proposal and initial prototype. In *Proc. ACM SIGCOMM VISA Workshop* (Barcelona, Spain, August 2009), ACM, pp. 63–72.

[114] Schwartz, B., Jackson, A. W., Strayer, W. T., Zhou, W., Rockwell, R. D., and Partridge, C. Smart packets for active networks. In *Open Architectures and Network Programming Proceedings, 1999. OPENARCH '99. 1999 IEEE Second Conference on* (1999), IEEE, pp. 90–97.

[115] Singh, A. An introduction to virtualization. `http://www.kernelthread.com/publications/virtualization/`, 2012.

[116] S.Kent, R. A. Security architecture for the internet protocol. RFC 2401, November 1998.

[117] Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw. 12*, 1 (2004), 2–16.

[118] Strachey, C. Time sharing in large fast computers. In *Proceedings of the International Conference on Information processing, UNESCO* (1959).

[119] Strassner, J. Den-ng: Achieving business-driven network management. In *Proc. IEEE/IFIP NOMS* (2002).

[120] Van der Ham, J., Dijkstra, F., Travostino, F., Andree, H., and de Laat, C. Using rdf to describe networks. *Elsevier FGCS 22*, 8 (2006).

[121] van der Ham, J., Grosso, P., van der Pol, R., Toonk, A., and de Laat, C. Using the network description language in optical networks. In *Proc. IFIP/IEEE ISINM* (2007).

[122] Villela, D., Campbell, A. T., and Vicente, J. Virtuosity: Programmable resource management for spawning networks. *Computer Networks, Special Issue on Active Networks* (2001).

[123] Vinoski, S. Corba: integrating diverse applications within distributed heterogeneous environments. *Communications Magazine 35*, 2 (1997).

[124] Introducing VMware virtual platform, 1999. Technical white paper - http://www.VMware.com.

[125] Realvnc website, 2012. `http://www.realvnc.com/`.

[126] Wang, Y., Keller, E., Biskeborn, B., van der Merwe, J., and Rexford, J. Virtual Routers on the Move: Live Router Migration as a Network-Management Primitive. *SIGCOMM CCR 38*, 4 (2008), 231–242.

[127] Website. CPLEX, June 2010. `http://www.cplex.com`.

[128] Website. SCIP, June 2010. `http://scip.zib.de`.

[129] Website. GUROBI, Apr. 2012. `http://www.gurobi.com`.

[130] Website. lpsolve, Apr. 2012. `http://sourceforge.net/projects/lpsolve/`.

[131] Website. SoPlex, Apr. 2012. `http://soplex.zib.de`.

[132] Weiser, P., and Nuechterlein, J. *Digital Crossroads: American Telecommunications Policy in the Internet Age*. MIT Press, 2005.

[133] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks.

[134] Wolsey, L. *Integer Programming*. Wiley-Interscience, New York, 1998.

[135] Wundsam, A., Mehmood, A., Feldmann, A., and Maennel, O. Network troubleshooting with mirror vnets. In *GLOBECOM Workshops (GC Wkshps)* (2010), IEEE.

[136] Yi, K., and Zhang, Q. Multi-dimensional online tracking. In *Proc. SODA* (2009), pp. 1098–1107.

[137] Yu, M., Yi, Y., Rexford, J., and Chiang, M. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM CCR 38*, 2 (2008).

[138] Zaheer, F., Xiao, J., and Boutaba, R. Multi-provider service negotiation and contracting in network virtualization. In *Proc. 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)* (2010).

[139] Zhang, Q., Xiao, J., Gürses, E., Karsten, M., and Boutaba, R. Dynamic service placement in shared service hosting infrastructures. In *NETWORKING* (2010), pp. 251–264.

[140] Zhu, Y., and Ammar, M. H. Algorithms for assigning substrate network resources to virtual network components. In *Proc. IEEE INFOCOM* (2006).

[141] Zhu, Y., Zhang-Shen, R., Rangarajan, S., and Rexford, J. Cabernet: Connectivity Architecture for Better Network Services. In *Proceedings of ReArch '08* (2008), ACM.