Decentralized Sensor Fusion using Periodic Peer-to-Peer Hypercube Gossiping

vorgelegt von
Dipl.-Ing. Tech. Inf.
Philipp Berndt
aus Berlin

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr.-Ing. Adam Wolisz
Gutachter:	Prof. Dr. Odej Kao
	Prof. Dr.-Ing. Reinhard German
	Prof. Dr.-Ing. Peter Gober

Tag der wissenschaftlichen Aussprache: 29. Oktober 2012

Berlin 2012

D 83

Acknowledgments

This thesis would not have been possible without the help of many people. Above all, I want to thank my supervisor Odej Kao for providing me with support and guidance whenever I needed it most and for giving me the chance to work with the great people at the CIT. Also, I would like to express my gratitude towards Reinhard German and Peter Gober for agreeing to review my thesis.

I am grateful for many fruitful discussions, especially to Paul McCabe for starting me off on this endeavor, and Felix Heine and Dominic Battré for priming me on P2P systems and warmly welcoming me to the CIT. I thank the entire working group for all the great time together, particularly Daniel Warneke, Andre Höing, Björn Lohrmann, Andreas Kliem, and Alexander Stanik, who have also been so kind as to proofread my thesis.

Many thanks go to Scene, LLC d/b/a Ookla and Hanna Lane for kindly providing me with global `speedtest.net` raw data.

Especially, I thank my girlfriend Saskia and my family for their great love, support, patience, and food.

Also, I want to thank the creators of all the excellent open source tools that helped me in compiling this thesis, particularly Linux, gcc, LaTeX, LyX, R, and X_{fig}. Finally, I want to thank Rob Hubbard and Martin Galway for creating indelible dreams, and Markus Behlau, Marcel Donné, Visa Röster, mind.in.a.box and many others for keeping them alive and my spirit up on my way through that endless maze.

Zusammenfassung

Unter Datenfusion versteht man das Aggregieren von Informationen aus mehreren Quellen mit dem Ziel, ein Ergebnis zu erhalten, welches aussagekräftiger ist als Informationen von einer einzelnen Datenquelle. Die Fusion von Sensordaten-Livestreams ist durch die fortlaufende Aggregation von Messdatenreihen gekennzeichnet. Ihre Anwendungen reichen von Audiokommunikation über die Überwachung von Geschäftsprozessen und das Verfolgen von Objekten bis hin zu gemeinsam nutzbaren haptischen virtuellen Umgebungen. Alle diese Anwendungen erfordern eine extrem niedrige Latenz. Dementsprechend ist die effiziente Skalierung bei begrenzter Datenbandbreite von entscheidender Bedeutung. Darüber hinaus fällt in Disseminationsnetzen ein erheblicher Latenzanteil auf Wartezeiten, die ihrerseits von der algorithmischen Implementierung des abstrakten Kommunikationsschemas abhängen. Vorhandene Aggregationsnetze sind entweder zentralisiert, vernachlässigen die Latenz oder setzen eine homogene Infrastruktur voraus. Dementsprechend wird die Komplexität von Kommunikationsalgorithmen allgemeinhin nur in Form von synchronen Kommunikationsrunden gezählt. Genauere Abschätzungen der Latenz oder Querzeiten sind schwer zu bestimmen.

Im Rahmen dieser Dissertation präsentiere ich einen neuen Ansatz zur dezentralisierten, bandbreiteneffizienten und latenzarmen periodischen Messdatenaggregation in heterogenen Umgebungen. Durch den Einsatz von Radix-r-Dezimation unter P2P-Knoten, die zu einem hyperkubischen Netz verbunden sind, wird in Bezug auf die Anzahl der Knoten ein logarithmischer Berechnungs- und Kommunikationsaufwand erzielt. Begleitet wird der Ansatz durch mehrere Methoden zur Reduzierung von Wartezeiten und zur analytischen Berechnung von Grenzen, Erwartungswerten sowie Verteilungen von Latenzen und Querzeiten. Die Leistungsfähigkeit der vorgestellten Methoden wird mithilfe von Simulationen untermauert. Das den Simulationen zugrunde liegende Netzwerkmodell basiert auf umfangreichen Netzwerkmessungen, unterstützt durch zwei neue Ansätze: zum einen zur repräsentativen und doch effizienten Latenzmodellierung in P2P-Netzen und zum anderen zur stationären Emulation periodischer Kommunikation. Komplettiert werden die Ansätze durch die Erörterung der Implementierung zweier beispielhafter Sensordatenfusionsanwendungen, nämlich skalierbare Audiokommunikation für virtuelle Umgebungen mit vielen Teilnehmern und skalierbare Datenfusion zur Objektverfolgung.

Abstract

Fusion marks the aggregation of information from multiple sources, yielding a result that is more valuable than information from any source alone. The fusion of live sensor streaming data is characterized by the ongoing aggregation of series of measurements. It has applications in areas ranging from audio communications, to business process monitoring, to live object tracking to shared haptic virtual environments. All of these applications require extremely low-latency, so efficient scaling becomes an issue in the face of limited bandwidth. Moreover, in dissemination networks, a considerable portion of latency can be traced back to sojourn times, which depend on the algorithmic implementation of the high-level communication scheme. Existing aggregation networks are centralized, latency insensitive, or assume a homogeneous infrastructure. Accordingly, complexity of communication algorithms is commonly only considered as the number of synchronous communication rounds. Accurate estimates for the latency or traversal time to be expected are hard to come by.

In this dissertation I will present a novel approach to the decentralized, bandwidth-efficient and low-latency aggregation of high-frequency periodic measurements in heterogeneous environments. By employing radix-r-decimation among peers interconnected into a hypercube network, a logarithmic computation and communication effort is achieved with regard to the number of nodes. The approach is accompanied by several methods for the reduction of wait latency as well as the analytical computation of bounds, expectations, and distributions of latency and traversal time. The performance of the presented methods is corroborated by simulations. The network model underlying the simulations is based on large-scale network measurements, aided by two new approaches for the representative yet efficient modeling of latency in peer-to-peer networks as well as the steady-state emulation of periodic communication. The approaches are supplemented by detailed considerations of two exemplary sensor fusion applications, namely scalable audio communication for massively multi-user virtual environments and scalable decentralized data fusion for object tracking.

Contents

Chapter 1

Introduction

Contents

One of the key tasks in time-sensitive information processing is the efficient aggregation of data from multiple sources: Apart from the problem of transferring all information to one place, the information must ultimately be summarized to be of practical use. Any decision making process relying on multiple information sources comprises the derivation of some sort of aggregate or *Entscheidungsvorlage*[1]. Depending on the application, the summary or aggregation operation differs. Simple generic aggregation functions include total sum, arithmetic mean, minimum, maximum, median and other quantiles.

For non-trivial amounts of data or more complex aggregation functions, the computation of the aggregate can be costly. Fortuitously, many aggregation functions lend themselves to parallelization naturally; so performing the aggregation in a parallel fashion can in fact decimate execution time. Given that the information sources are distributed already, it stands to reason to include the computing nodes where the information originates in the aggregation process—coupled with the likely availability of processing power and communication capability in sensor nodes [69]. [98] demonstrates how such *in-network-execution* can yield an order of magnitude reduction in communication compared to centralized approaches. The authors propose an aggregation tree, a tree rooted at some base station, wherein every node aggregates data it receives from its children and passes the partial result up to its parent. As data flows up this tree, it is aggregated according to the aggregation function and flows out at the root.

If, however, the final result is needed at the source nodes, it must, again, be disseminated (*broadcast*) to all interested recipients. Obviously, such a two-stage scheme is non-optimal. Apart from the doubled path lengths, which lead to increased latency, all communication relies on the root node as a central component. If this root node fails, the provision of the aggregate is disrupted for all nodes. Furthermore, the communication scheme is asymmetric or "unfair", as some nodes have longer paths than others and thus will always receive the aggregate later. As postulated in [57], completely decentralized execution bears several advantages: The abolishment of central components such as

[1] Executive summary of all information relevant for making a decision

central nodes, common communication facilities, or total knowledge of network topology ensures scalability by absence of central bottlenecks as well as modularity and flexibility by obviating the need for knowledge of the network. Effectively hyped around the turn of the millennium, this communication paradigm of decentralization has since then received much attention under the name of *peer-to-peer (P2P)* communication. In P2P aggregation there is no distinguished root node. Instead, every node's contributing information is to be communicated to all other nodes. This elemental information dissemination problem is referred to as *gossiping*, also known as *total exchange*. It is distinct from *broadcasting,* where content is disseminated from only *one* source to all consumers. Gossiping is inherent to symmetric distributed systems and parallel and distributed computing and has thus received due attention [9].

The *classic gossip problem*, formulated in Subsection 2.1.3, describes a one-time dissemination. In typical sensor applications, however, data is sampled continuously, periodically. Consequently, the required communication is also not singular but comprises periodic sending of data packets, usually containing several measurements or *samples,* to reduce the packet overhead. This recurring communication task of live streaming between peers will be the focus of this thesis and is subsequently referred to as *periodic gossiping*.

The applications for distributed aggregation of live streaming data are multifarious. One intuitive example is audio conferencing (Subsection 2.6.1), where the sensors are microphones, the streams are series of audio samples and the aggregation function is the sum over all sources. However, live streaming is not limited to media streaming. Another featured application is scalable decentralized data fusion for live object tracking (Subsection 2.6.2). Further possibilities include real-time business intelligence [20], agent based management for smart grids [55], and shared haptic virtual environments[2].

1.1 Problem Brief

Most applications comprising real-time fusion of live streaming data from multiple sources have demanding latency requirements. Scalability becomes an issue in the face of scarce bandwidth. Consequently, an efficient dissemination network and communication scheme is called for.

In dissemination networks, a considerable portion of latency can be traced back to sojourn or bide time [96], denoting the time information spends waiting after arriving at one node and before being relayed to another. The algorithmic implementation of the high-level communication scheme has a strong impact on this wait delay and should be constituted as so to keep it to a minimum. Whereas great efforts have been made to reduce the gossip complexity, the communication schemes generally do not consider wait latency.

Moreover, accurate estimates for the latency or traversal time to be expected are rarely available. Prevalent complexity measures [27], communication schemes [82], or schedulers [66] only regard time as integer number of rounds. Multiplying this number with the sum of the maximum network delay and the period of a round only yields a trivial upper bound on the total latency. Although gossiping belongs to the most investigated communication problems [9], no more accurate time bounds have been published.

[2]The goal of haptic rendering is to enable a user to touch, feel, and manipulate virtual objects through a haptic interface [7]. Shared haptic virtual environments (SHVE) enable multiple users to collaboratively perform tasks in environments considered inaccessible, dangerous or even of incompatible scaled, such as underwater and on-orbit servicing, bomb disarming, radioactive material manipulation, microsurgeries and minimally invasive medical procedures [31, 24].

Using simulations to derive results is problematic, too. Prevalent latency models are either overly simplistic, require sophisticated configuration, or are prohibitively inefficient for simulating streaming communication, where every node spouts packets with high frequency.

The problem addressed by this thesis can thus be summarized as the problem of providing decentralized bandwidth-efficient low-latency aggregation of high-frequency periodic measurements in heterogeneous environments and with predictable latency, along with a faithful yet efficient means for simulation.

1.2 Contribution

The contribution of this dissertation is divided into two major contributions, two auxiliary contributions, and several minor practical contributions:

The first major contribution is a novel approach to the scalable aggregation and dissemination of streaming data. This approach, presented in Chapter 3, makes use of radix-r-decimation among peers interconnected into a hypercube network and exhibits a logarithmic computation and communication effort with regard to the number of nodes.

On this basis, the second major contribution, presented in Chapter 6, comprises several methods for the reduction of wait latency as well as the analytical computation of bounds, expected values, and distributions of wait latency and traversal time.

As auxiliary contributions that were needed for the evaluation of the latter but are also of independent use, I present approaches and methods for the efficient modeling of latency in P2P networks (Chapter 4) as well as for emulating and visualizing periodic communication (Chapter 5).

These contributions are supplemented by practical considerations regarding the implementation of two exemplary sensor fusion applications, namely scalable audio for massively multi-user virtual environments in Chapter 8 and scalable decentralized data fusion for object tracking in Chapter 9.

Parts of this thesis have been published in the following publications:

1. **Philipp Berndt**
   Minimizing Wait Latency in Periodic P2P Hypercube Gossiping
   In: Proceedings of the 20$^{\text{th}}$ Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP 2012), IEEE Computer Society, pp. 341–345, 2012

2. **Philipp Berndt, Matthias Hovestadt, Odej Kao**
   Characterizing Latency in Periodic P2P Hypercube Gossiping
   In: Proceedings of the Fourth International Conference on Communications Systems and Networks (COMSNETS 2012), IEEE, pp. 1–8, 2012

3. **Philipp Berndt, Alexander Stanik**
   Scalable Decentralised Data Fusion using Hypercube Gossiping
   In: Proceedings of the Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2011), IEEE, pp. 502–507, 2011

4. **Philipp Berndt, Odej Kao**
   Time Bounds for Periodic Hypercube Gossiping

In: Proceedings of the 1st International Workshop on Worst-Case Traversal Time (WCTT 2011), ACM, pp. 19–26, 2011

5. **Philipp Berndt, Matthias Hovestadt, Odej Kao**
Crowd Buzz: Scalable Audio Communication for MMVEs using Latency Optimized Hypercube Gossiping
In: Proceedings of the IEEE International Symposium on Audio-Visual Environments and Games (HAVE 2011), IEEE, pp. 182–187, 2011

6. **Philipp Berndt, Matthias Hovestadt, Odej Kao**
REEL: Modeling, Simulating and Visualizing Periodic Peer-to-Peer Communication
In: Proceedings of the 2nd International Conference on Computer Modelling and Simulation (CSSim 2011), FIT, Brno University of Technology, pp. 19–26, 2011

7. **Philipp Berndt, Martin Raack, Odej Kao**
Spanning a Global Delay Model from HTTP-Measurements
In: Proceedings of the Annual International Conference on Network Technologies & Communications (NTC 2010), Global Science and Technology Forum (GSTF), pp. N77–N82, 2010

8. **Philipp Berndt, Dominic Battré, Odej Kao**
A Hybrid Approach to Modeling End-to-End Delay in P2P Networks
In: Proceedings of the 2010 ACM Workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Networking (AVSTP2P 2010), ACM, pp. 37–42, 2010

9. **Philipp Berndt**
Using Symmetric Distributed Processing for Peer-to-Peer VoIP Conferencing in Auditory Virtual Environments
In: Proceedings of the 7th International Workshop on Peer-to-Peer Systems (IPTPS 2008), USENIX Association Berkeley, pp. 1–6, 2008

1.3 Outline of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2: Background and Problem Definition provides some theoretical background on information dissemination, live streaming, aggregation, the principal of P2P systems, hypercubes and related topologies, and the exemplary applications. On this basis, the problem statement is refined.

Chapter 3: Periodic Peer-to-Peer Hypercube Gossiping develops, on the basis of some assumptions and requirements, firstly, the network topology and dynamics, and secondly, the communication and aggregation schemes for periodic hypercube gossiping. The former part deals with the structural properties of the network, the problem of dynamically assigning nodes to network positions, subject to specific constraints, as well as deviations from the perfect hypercube. The latter part defines which nodes exchange messages, the contents of these messages, and the computation of the aggregate.

Chapter 4: Modeling Latency in Peer-to-Peer Networks develops and parametrizes an efficient latency model with due regard to the Internet anatomy, as will be required for the evaluations that follow.

Chapter 5: Emulating and Visualizing Periodic Communication demonstrates how timing and latencies of the network can be modeled and visualized in a way that allows for quick apprehension and presents the tools used to interactively manipulate the model, study interdependencies, and efficiently conduct experiments.

Chapter 6: Minimizing Latency points out the need, potential, and means to minimize latency in periodic hypercube gossiping. It examines wait latency and presents five timing modes along with maximum and expected latencies. Analytic measures are corroborated with simulation results.

Chapter 7: Implementation presents a realization of the approach, including a flexible yet efficient architecture and an extensible protocol.

Chapter 8: App. 1: Scalable Audio Communication for MMVEs shows how periodic P2P hypercube gossiping can be used to facilitate auditory virtual environments (AVE), i. e. virtual environments where people can hear each other more or less like in real environments, particularly massively multiuser virtual environments (MMVE) that accommodate a high number of participants.

Chapter 9: App. 2: Decentralized Data Fusion for Object Tracking demonstrates how periodic P2P hypercube gossiping allows for scalable decentralized data fusion as is required for live object tracking.

Chapter 10: Conclusion concludes the thesis with a summary of the contributions and potential extensions.

Chapter 2

Background and Problem Definition

Contents

The goal pursued in this thesis, as laid out in the previous chapter, is to find a solution for the efficient fusion of streaming data from multiple sources. "Efficient fusion", is to describe a solution that for a high number of nodes with finite bandwidth fuses the information in minimal time. In this sense, a naïve solution would have all nodes send their data to one node that does the aggregation. Obviously, neither the reception of data from all other nodes nor the computation of the aggregate on one node scales: The effort

of the former is $\mathcal{O}(n)$ and the effort of the latter may be even higher, depending on the aggregation function.

As a second, inefficient example consider arranging all nodes in a line, i. e. path topology, and have them pass the information from node to node. Obviously, this solution does not scale either, as the time complexity until all information reaches the last node is $\mathcal{O}(n)$.

Efficient computation schemes that recursively break a problem into two or more parts until it is easily solvable may have been known as long as since 300 BC[1] and are today subsumed under the term *divide-and-conquer algorithm*[2]. In [109] the divide-and-conquer

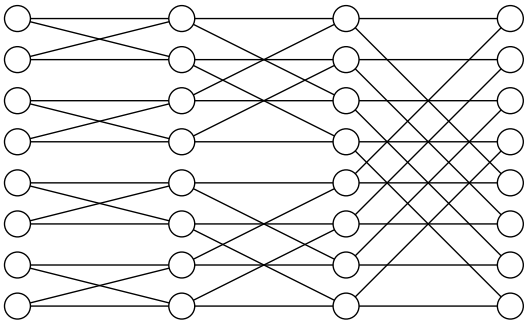


Figure 2.0.1: Butterfly Graph BF_3

approach is called "one of the most successful programming paradigms. Especially in the field of parallel processing one of the most natural methods to solve problems is to divide them into subproblems (*divide-step*), solve these in parallel (*recursion*), and then recombine the solutions to a solution of the given problem instance (*conquer-step*)." For those divide-and-conquer problems for which the total size of the subproblems on any level of the recursion does not exceed the parent problem size an optimal generic implementation on hypercubes is presented.

Early hardware realizations of the hypercube topology in the form of expensive parallel computers were undertaken in the 1980s. One of the first notable systems was the Caltech Cosmic Cube [136]. Development started with a 4-node, Intel 8086 CPU, 8087 floating-point coprocessor, 128KB RAM based prototype in 1981 and culminated 1989 in the Mark IIIfp, employing 128 nodes à two Motorola 68020 CPUs, a Weitek XL floating point chipset, 4MB RAM and delivering 500 MFLOPS [53]. The 64-node was soon followed by the commercial hypercube systems, such as the Intel iPSC/1 introduced in 1985, and consisting of 32 to 128 nodes with each a 80286 CPU with 80287 math coprocessor and

[1] The Inakibit-Anu Babylonian tablet contained a list of roughly 500 sorted reciprocals and is the earliest known example of a large file that is in lexicographic order. "And this is one of the reasons his work is so impressive, as anyone who has tried to sort 500 cards by hand will attest. To get some idea of the immensity of this task, consider that it takes many hours to sort 500 large numbers by hand nowadays; image how difficult it must have been to do this job in ancient times. Yet Inakibit is likely to have done it, since there is no obvious way to generate such a table in order." [83]

[2] One particularly notable specimen is an algorithm for the efficient evaluation of coefficients of Fourier series that was discovered in an unpublished manuscript dated 1805 AD and written by Carl Friedrich Gauss, who has applied it to interpolate the trajectories of the asteroids Pallas and Juno. The algorithm had gone mostly unrecognized until it was one century later rediscovered by J. W. Cooley and J. W. Tukey [62] and is ever since known as the *Cooley-Tukey Fast Fourier Transform* (*FFT*) algorithm [37]. The structure of the algorithm's data flow resembles the shape of a butterfly and is hence called butterfly graph. It is shown in Figure 2.0.1. This data flow, embedded in a hypercube network, is also exploited in the course of this work to facilitate efficient sensor fusion through parallelization.

512K of RAM [75] and the nCube 10 [61], designed by Intel employees and released in 1985, supporting 1024 nodes with custom modules à 128 KByte RAM and 2 MIPS.

The advent of commodity PCs in conjunction with affordable Internet connectivity has paved the way for a new reality of interconnected computers that everyone could be part of. Although *peer-to-peer (P2P)* computing, as it was hyped about the turn of the millennium, was in public mostly known for file-sharing and the distribution of copyrighted material, it has, all the same, created a new paradigm of distributed and decentralized computing in heterogeneous networks that revolves around communication, cooperation, and collaboration. Increased robustness, scalability, and flexibility are some of its advantages. Several P2P networks employ hypercube organization or routing strategies [134, 121, 131, 153, 95, 97].

Since the conquer-step already comprises the aggregation of partial results into a final result, it stands to reason that also fusion applications can benefit from this approach. As will be shown below, even the harder problem of producing the aggregate at each node can be solved optimally on a hypercube topology. However, to make the P2P hypercube fusion approach viable, several problems related to restricted connectivity, bandwidth, latency, unreliable networks, node fluctuations, and data hygiene must be solved.

The remainder of this chapter covers some basic concepts that are used throughout this work, starting with the foundations of information dissemination to live streaming to data aggregation, the principles of P2P systems, the hypercube topology to two exemplary applications. Based on these foundations and terminatory to this chapter, the problem statement is defined.

2.1 Information Dissemination

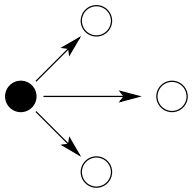
Information dissemination denotes a process by which information is spread to many recipients and without feedback or response [120]. In communications, it is used as a superordinate concept and describes several communication modes between multiple participants, namely *broadcasting*, *accumulation,* and *gossiping*. Throughout this work I use the terms and concepts defined in [65] and summarized below:

The *structure* or *topology* of an interconnection network is defined as a connected undirected graph $G = (V, E)$ where V is a finite set of vertices (*nodes*) corresponding to the processors and the edges in $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ corresponding to the communication links of the network.

A communication algorithm or *communication scheme* is a sequence of elementary communication steps called *communication rounds*.

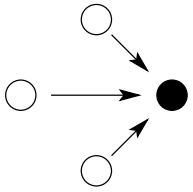
2.1.1 Broadcasting

Broadcast problem for a graph G and a node v of G Let $G = (V, E)$ be a graph and let $v \in V$ be a node of G. Let v know a piece of information $I(v)$ which is unknown to all nodes in $V \setminus \{v\}$. The problem is to find a communication strategy such that all nodes in G learn the piece of information $I(v)$. In other words, the broadcast problem refers to the problem of spreading the knowledge of one processor to all other processors in the network.



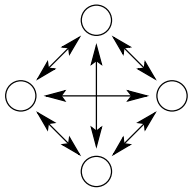
2.1.2 Accumulation

Accumulation problem for a graph G and a node v of G Let $G = (V, E)$ be a graph and let $v \in V$ be a node of G. Let each node $u \in V$ know a piece of information $I(u)$, and let, for any $x, y \in V$, the pieces of information $I(x)$ and $I(y)$ be "disjoined" (independent). The set $I(G) = \{I(w) \mid w \in V\}$ is called the cumulative message of G. The problem is to find a communication strategy such that the node v learns the cumulative message of G. In other words, the accumulation problem is the problem of accumulating the knowledge of all processors in one given processor.



2.1.3 Gossiping

Gossip problem for a graph G Let $G = (V, E)$ be a graph and let, for all $v \in V$, $I(v)$ be a piece of information residing in v. The problem is to find a communication strategy such that each node from V learns the cumulative message. In other words, the gossip problem describes the problem of accumulating the knowledge of all processors in each processor of the network.



The term *gossip complexity* denotes the necessary and sufficient number of communication rounds to complete the gossip problem, subject to one of the communication modes below.

2.1.4 Communication Modes

In the context of information dissemination, the *communication mode* defines the constraints of communication on a graph, i. e. the allowed concurrent communication operations within one round. In [86] modes are categorized according to *degree* of a feasible communication step and the *duplex mode* of communication channels. The degree describes the maximal number of simultaneous communication activities allowed at each node. It is commonly assumed to be either one or unrestricted. The corresponding modes are called *pairwise* or *simultaneous*, respectively. The duplex describes whether between a pair of connected nodes, the nodes can send and receive at the same time. If this is the case, the link is called *full-duplex*, otherwise *half-duplex*.

Full-duplex Simultaneous, aka. F* Mode

In this most powerful mode, in a single round, each node may both send and receive via all of its adjacent edges. A two-way simultaneous communication algorithm for a graph $G = (V, E)$ is defined as a sequence $E_1, E_2, ..., E_r$ of some sets $E_i \subseteq E$ of active edges. For every $i \in \{1, \ldots, r\}$, E_i is called a round in the two-way simultaneous mode.

Half-duplex Simultaneous, aka. H* Mode

In this mode, in a single round, each node may communicate with all of its neighbors, however, via each link it may either send or receive. A one-way simultaneous communication algorithm for a graph $G = (V, E)$ is defined as a sequence $E_1, E_2, ..., E_r$ of sets $E_i \subseteq \overline{E}$, where $\overline{E} = \{(v \to u), (u \to v) \,|\, (u, v) \in E\}$ and $(x \to y) \in E_i$ for some $i \in \{1, \ldots, r\}$ implies $(y \to x) \notin E_i$. For every $i \in \{1, \ldots, r\}$, E_i is called a round in the one-way simultaneous mode.

Full-duplex Pairwise, aka. F1, Two-way, or Telephone Mode

In this mode, in a single round, each node may both send and receive via one of its adjacent edges. In [65], a two-way [pairwise] communication algorithm for a graph $G = (V, E)$ is defined as a sequence $E_1, E_2, ..., E_r$ of some sets (matchings) $E_i \subseteq E$ where, for each $i \in \{1, \ldots, r\}$ and all $(x_1, y_1), (x_2, y_2) \in E_i$, $\{x_1, y_1\} \neq \{x_2, y_2\}$ implies $\{x_1, y_1\} \cap \{x_2, y_2\} = \varnothing$. For every $i \in \{1, \ldots, r\}$, E_i is called a round in the two-way [pairwise] communication mode.

Half-duplex Pairwise, aka. H1, One-way, or Telegraph Mode

In this mode, in a single round, each node may either send or receive via one of its adjacent edges. In [65], a one-way [pairwise] communication algorithm for a graph $G = (V, E)$ is defined as a sequence $E_1, E_2, ..., E_r$ of sets $E_i \subseteq \overline{E}$, where $\overline{E} = \{(v \to u), (u \to v) \,|\, (u, v) \in E\}$ and if $\{(x_1 \to y_1), (x_2 \to y_2)\}$ are two distinct elements of E_i for some $i \in \{1, \ldots, r\}$, then $\{x_1, y_1\} \cap \{x_2, y_2\} = \varnothing$. For every $i \in \{1, \ldots, r\}$, E_i is called a round in the one-way [pairwise] mode.

Systolic Modes

Based on the notion of systolic systems [87], in which data is pumped rhythmically and synchronously through a hard-wired processor network, more or less like blood through the body, [65] defines a *p-systolic* algorithm as a communication algorithm $E_1, E_2, ..., E_m$ where $E_i = E_{i+p}$ for all $1 \leq i \leq m - p$. $E_1, E_2, \ldots, E_p$ is called the period/cycle of algorithm and p the period length.

Periodic Modes

A p-systolic algorithm is said to be *p-periodic* if, for every edge $e \in E$, e is active in at most one round [65].

2.2 Live Streaming

Live streaming is often thought of as being all about video or audio. Rather, live streaming can be more generally thought of as a process in which one or more sensors are used

to repeatedly obtain readings that are transported step-by-step over a communication network to one or more receivers. This includes all kinds of sensor applications. But also in monitoring applications, to give another example, a stream of values is often more useful than a single isolated value [98].

One precursory application to live streaming is analog telephony: Using microphones, sound waves, i. e. variations in air pressure, are transformed into variations in voltage, amplified and conducted via wires to the speaker on the receiving side, which transforms them back into sound waves. In analog telephony, however, there is no stream of samples but rather a continuous signal that changes over time.

By contrast, in digital telephony, such as ISDN [70, 71], a quantization is done in time and displacement. An analog-to-digital converter (*ADC*) converts the continuous signal into a discrete time digital representation. The samples are encoded and streamed across the network.

To stream samples across packet switched networks, such as the Internet, the sample values must be enveloped in various packet structures such as UDP and IP. Thus, the transmission of each packet induces a data overhead due to various headers as well as associated processing effort for labeling and checksumming. For high sampling rates, sending data packets with these sampling rates would in many cases overload both the processing node and the network link. As a consequence, high frequency streaming data is usually not sent as separate samples but in chunks of many consecutive samples. This chunking also facilitates the use of more efficient compression, such as quantization in the frequency domain or, for instance, with the aid of psycho-acoustic models in the case of audio data. The rate with which packets are sent by one node to each of its neighbors will be called *send rate*; its reciprocal is called *(send) period* T. The send rate is usually chosen to be a divisor of the sampling frequency to yield an integer number of samples per data packet. Thus, the case in which packets containing just one sample each are sent with f_{samp} is a special case.

To summarize, the periodic transmission of data packets containing a time series of samples is a common characteristic for streaming applications. As this has strong implications on latency, it will be regarded extensively in Chapter 6 of this work.

2.3 Aggregation

The capability to quickly extract useful information from large amounts of data has been a vital cognitive task since primeval times. Accordingly, most fast moving animals possess dedicated neural networks that preprocess visual data and produce useful aggregates, i. e. visual patterns such as edges or movements. It is the aggregates, not the raw data, that are eventually used to control and plan action [25]. But also for mankind the deliberate aggregation of measurements soon became important. Data acquisition and book-keeping date back to the Uruk period (ca. 4000 to 3100 BC), along with the invention of writing and thus history itself [135]. Since then the amount of data has grown vastly, accelerated by inventions such as the printing press, computers, and the Internet. In the year 2007 AD, the digital universe, i. e. the amount of information that was created, captured, or replicated, exceeded 281 exabytes [54]. With the amount of digital information now roughly quadrupling every three years, it is apparent that efficient aggregation will remain an important issue in the future.

In Section 2.1 the basic communication tasks, *broadcasting*, *accumulation,* and *gossiping* were stated. For the tasks of *accumulation* (2.1.2) and *gossiping* (2.1.3) the cumulative

message does not necessarily comprise all the raw sensor readings. In fact, the communication of the entirety of measurements is often completely unfeasible. Many times, however, a summary of the information is all that is needed. By way of *in-network aggregation*, data volume can be reduced in the field. Instead of raw measurement data, the nodes communicate *partial aggregates*.

[98] proposes an approach to in-network stream aggregation in low-power wireless sensor networks using an aggregation tree. Every node combines the data it receives from its children with its own and sends the partial aggregate to its parent. The final aggregate flows out at the root node.

In order to generically support a wide variety of aggregates, aggregation is typically broken into three abstract functions, as stated in [98]:

- a merging function f,
- an initializer i, and
- an evaluator e.

In general, f has the following structure:

$$< z >= f(< x >, < y >)$$

where $< x >$ and $< y >$ are multi-valued *partial state records*, computed over one or more sensor values, representing the intermediate state over those values that will be required to compute an aggregate. $< z >$ is the partial state record resulting from the application of function f to $< x >$ and $< y >$. For example, if f is the merging function for AVERAGE, each partial state record will consist of a pair of values: SUM and COUNT, and f is specified as follows, given two state records $< S_1, C_1 >$ and $< S_2, C_2 >$:

$$f\,(< S_1, C_1 >, < S_2, C_2 >) =< S_1 + S_2, C_1 + C_2 >$$

The initializer i is needed to specify how to instantiate a state record for a single sensor value; for an AVERAGE over a sensor value of x, the initializer $i(x)$ returns the tuple $< x, 1 >$. Finally, the evaluator e takes a partial state record and computes the actual value of the aggregate. For AVERAGE, the evaluator $e(< S, C >)$ simply returns S/C.

In [98, 56] aggregates are classified according to their tolerance of loss, duplicate sensitivity, monotonicity, and particularly, amount of state[3] required for each partial state record:

> For example, a partial AVERAGE record consists of a pair of values, while a partial COUNT record constitutes only a single value. ...
>
> - In *Distributive* aggregates, the partial state is simply the aggregate for the partition of data over which they are computed. Hence the size of the partial state records is the same as the size of the final aggregate.
> - In *Algebraic* aggregates, the partial state records are not themselves aggregates for the partitions, but are of constant size.
> - In *Holistic* aggregates, the partial state records are proportional in size to the set of data in the partition. In essence, for holistic aggregates no useful partial aggregation can be done, and all the data must be brought together to be aggregated by the evaluator.

[3] i. e. data volume to be transmitted

- *Unique* aggregates are similar to holistic aggregates, except that the amount of state that must be propagated is proportional to the number of distinct values in the partition.
- In *Content-Sensitive* aggregates, the partial state records are proportional in size to some (perhaps statistical) property of the data values in the partition. Many approximate aggregates proposed recently in the database literature are content-sensitive. Examples of such aggregates include fixed-width histograms, wavelets, and so on; see [6] for an overview of such functions.

All but the holistic aggregates yield a reduction in data volume, particularly distributive and algebraic aggregates, resulting in significantly lower bandwidth requirements compared to centralized aggregation.

2.4 Peer-to-Peer Systems

2.4.1 Characterization

Peer-to-peer (P2P) systems are characterized as distributed systems in which nodes of equal roles and capabilities exchange information and services directly with each other [151]. In contrast to the conventional client-server model, all nodes perform the same functions cooperatively but autarkically. Among the nodes, there is no distinction between clients and servers.

2.4.2 Advantages

The main advantages of P2P systems are robustness, scalability, adaptiveness, and cost effectiveness.

A centralized system is very brittle, as the destruction of just one node, a *single point of failure,* can knock out the whole system. As opposed to this, depending on the specific topology, it takes the destruction of a considerable number of nodes to completely disable a decentralized system [5].

But a central component does not even have to fail to cause system dysfunction. If the number of requests rises, either because of actual demand or because it is being targeted by a DOS attack[4], the central component will easily become a bottleneck and may effectively grind the whole system to a halt. In consideration thereof, to attain scalable data fusion in sensor networks, [57] stipulates the abolishment of central components such as single fusion centers, central nodes, common communication facilities, or total knowledge of network topology, thus laying the foundations for scalable P2P systems.

Whereas conventional client-server systems are static and changes to the system generally incur service downtime, P2P systems are highly dynamic and adaptively scale by simply adding more nodes to the network. All the same, individual nodes can be removed from the network without causing a disruption of the system.

Finally, P2P approaches are often cheaper because cheap and unreliable equipment can be used [5] or costs are even carried entirely by the public running the nodes.

[4] A Denial Of Service attack comprises sending a high number of requests to a server in order to overload it.

2.4.3 Applications

P2P systems are used for multifarious applications. The most popular domain is that of content distribution with applications such as file sharing, content delivery networks (CDNs), software distribution, and media streaming.

In contrast, the applications relevant to this work are in the fields of real-time communications and distributed computing. Since the 1980s, interest in parallel and distributive computing has been growing rapidly. While it was initially concerned with the design of parallel architectures, parallel algorithms, and expensive parallel computers, it has since then moved more and more to the world of interconnected computers, namely grid computing, cloud computing, and P2P computing. Especially, the last-named enables us to construct distributed supercomputers exceeding the computing power of any single supercomputer or computing center [10]. Notable P2P or public resource computing platforms include BOINC [3] and GTapestry/P2HP-2 [73], which are used, for instance, in search for extraterrestrial intelligence[5], climate studies, astrophysics, epidemiology, and protein structure prediction.

2.4.4 Mode of Operation

Setup

In P2P systems, not only the operation but also the setup of the distributed network is to take place without reliance on central components. A P2P system configures itself dynamically using *self-organization*. Nodes may join or leave the network at any time. An arriving node must find a suitable position in the network with just a little[6] help from its initial contacts.

Communications

In so-called *systolic* computations, commonly used for parallel computing in VLSI architectures, the whole network works synchronously and data is pulsed through the network like blood through the human body. The processors are Moore machines, where the output is driven by registers. In *Half-systolic* computations, the processors are Mealy machines, where the output emanates from either registers or arithmetic-logic units (ALUs).

By contrast, P2P systems feature neither systolic nor half-systolic communications. Not only the lack of a global clock but also asymmetric network delays, which vary greatly for different source/destination pairs, render synchronous communication if not impossible then at least undesirable, as we will see in Chapter 6.

One essential characteristic of typical P2P systems is the obligatory capability to cope with typical consumer Internet access links. These are characterized by a considerable access delay and a low bandwidth in the upstream direction. Disregarding this fact in either design or evaluation will yield poor performance or false results, respectively. For this reason Chapter 4 is devoted to faithfully and efficiently modeling delay in P2P networks.

Topologies

Regarding their topology, two fundamental kinds of P2P systems are discerned, *unstructured* and *structured* networks.

[5] `http://setiathome.ssl.berkeley.edu/`

[6] i. e. using just a small, e. g. logarithmic, number of messages with regard to total number of nodes

The first publicly received decentralized P2P network was GNUTELLA [130], an unstructured network, commonly used for file sharing. New nodes would attach to so-called *supernodes*, having extended responsibilities. This resulted in few nodes having connections to many nodes, i. e. a power-law distribution of node degree. Locating a node with a particular item had to be done inefficiently and unreliably by flooding the neighborhood with search queries.

Today's P2P networks are predominantly structured networks. In these, every node has an overlay ID, thus facilitating addressing on the application layer. Messages can be routed efficiently to any named node. Additionally, content is organized in such a way as to be found efficiently, usually in the order of $\mathcal{O}(\log N)$, e. g. by using distributed hash tables (DHT) or search trees. The node degree is commonly expected to be in between $\mathcal{O}(\log N)$ and $\mathcal{O}(1)$.

Routing

Two disparate algorithms can be used for routing a message to a destination, *recursive routing* and *iterative routing*. In recursive routing the message is forwarded from node to node until it reaches the destination node. In iterative routing, the source node asks every intermediate node for directions until it can contact the final node. In an error-free environment, recursive routing is faster because it needs just half as many messages as iterative routing. However, the originator is kept in the dark regarding routing progress or whether the message has been lost on the way. As a compromise, in [88], *hybrid routing* is proposed, whereby at every step an additional message is sent back to the originator.

2.4.5 Challenges

Obviously, the advantages of P2P systems do not come without complications. P2P networks are complex distributed systems and their programming is non-trivial. As nodes may leave the network unforeseen, probabilistic assumptions must be used in many places where reliability is assumed ordinarily. Structural damage caused by churned nodes must be repaired and the system must be secured against (at least a minority of) malicious nodes.

NAT / Hole Punching One particular problem arises from the widespread use of network address translation (NAT). Whereas the original addressing scheme intended globally unique and addressable identifiers for every hosts connected to the Internet, the rapid expansion of the Internet has led to a scarcity of IP addresses that has made it attractive to share IP addresses among several hosts. At this, the public IP address is held by a NAT router to which the hosts are connected via a local area network with private addresses. Any in such a way "*natted*" or "*firewalled*" host wishing to communicate with a server sends its request packet to the NAT router. The NAT router replaces the packet's source address with its own and the source port with an arbitrary unused port number before forwarding the packet to the server. When it receives the reply from the server it re-substitutes its own socket address by that of the original host before delivering the packet. Any unsolicited packets arriving at the NAT router cannot be mapped to a particular host and are rejected. As a consequence, one natted host cannot contact another directly. To nevertheless establish a communication between the two of them, the hosts need to be introduced to each other by a third party that is already in contact with both of them. This procedure is referred to as "*hole punching*" [52]. At this, the third party informs each of the hosts about

the other's public socket address. Thereupon, both hosts attempt to contact the other using the communicated public socket addresses. It is only after the attempt of the local host that the attempt of the remote host can be put through. This complication needs to be regarded in all P2P communication.

Testability and Evaluation Another problem springs from the fact that design tests, debugging, and evaluation can generally only be performed using simulators. Running the simulations can cost a lot of time. Consequently, diligence should be exercised to employ an efficient underlay network model (such as the one presented in Chapter 4) and, where possible, use network emulation (as presented in Chapter 5) instead of discrete event simulation.

2.5 Hypercubes

In parallel and distributed computing, the performance of systems stands and falls with their ability to effectively distribute data between their nodes [65]. The hypercube is not only one of the most versatile and most efficient network topologies known, its regular structure also allows for very simple and efficient algorithms [17]. This makes it an exceptional choice for the architecture of a massively parallel system [92].

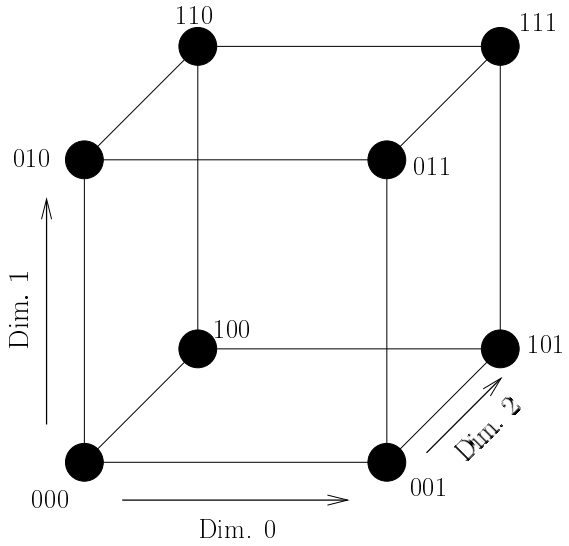


Figure 2.5.1: Binary hypercube H_3

2.5.1 Definition

A (binary) hypercube of dimension m, denoted H_m, is the graph whose nodes are all (binary) strings of length m and whose edges connect those strings that differ in exactly one position, called the *dimension* i of the edge. This way, the edges of a hypercube are partitioned in a natural way according to the dimension they traverse.

2.5.2 Properties

H_m has 2^m nodes, $m \cdot 2^{m-1}$ edges and diameter m. Each node has exactly degree m [66]. Figure 2.5.1 shows an illustration of the binary hypercube H_3, also called *3-cube,* along with binary node numbering.

Robustness

The hypercube has a high bisection width of $N/2$, i. e. it requires a removal of $N/2$ edges to split the hypercube into two disjunct networks. Alternatively, the hypercube can be bisected by removing a number of nodes in the order of $N/\sqrt{\log N}$, e. g. all nodes of Hamming weights $\left\lfloor \frac{\log N}{2} \right\rfloor$ and $\left\lceil \frac{\log N}{2} \right\rceil$ [92].

Symmetry

The hypercube is both node- and edge-symmetric. This means that any node can be mapped to any other by simply re-numbering the nodes. The same is true for the edges.

2.5.3 Gossiping on the Hypercube

For a hypercube of N nodes, all three of the fundamental communication problems (Section 2.1), namely broadcasting, accumulation and gossiping, can be solved in $\log_2 N$ communication rounds.

A two-way gossiping communication scheme according to Subsection 2.1.4 is the sequence

$$\{(0,1),(2,3),\cdots,(2^m-2,2^m-1)\},\cdots,\left\{\left(0,2^{m-1}\right),\cdots,\left(2^{m-1}-1,2^m-1\right)\right\},$$

e. g. for H_3 it consists of the three rounds (node IDs in binary notation)

$$\{(000,001),(010,011),(100,101),(110,111)\},$$
$$\{(000,010),(001,011),(100,110),(101,111)\},$$
$$\{(000,100),(001,101),(010,110),(011,111)\}.$$

Since there exists no graph on which the two-way gossip problem can be solved in less than $\lceil \log_2 N \rceil$ rounds, the scheme is optimal and H_m is a *minimal gossip graph* [66].

2.5.4 Variations on the Hypercube

Despite its many advantages, for some applications the hypercube is, in its original form, inapplicable. Therefore some variations of the hypercube were devised that pertain most of its favorable traits and give up a few others to make it more practical. Notable variations are presented below.

Incomplete and Compact Hypercubes

Sizes of ordinary hypercubes are restricted to powers of two. In [77] the construction of *incomplete hypercubes* is proposed for any number of nodes. In [17], using the definition of a *compact set* of nodes, the incomplete hypercube is generalized to that of n-node *compact hypercubes*.

Cube Connected Cycles

The degree of the hypercube H_m equals m, i. e. it grows with the size of the hypercube. This is problematic for the construction of arbitrary sized supercomputers from processors that will form nodes of the hypercube, as the number of links of the processors will have to be fixed beforehand. The *cube connected cycle* (CCC) graph [123] can be constructed from a hypercube by replacing each node with a cycle of length m. Thereby, the node degree becomes fixed to three.

Hierarchical Hypercubes

The hierarchical hypercube [107] forms a compromise between a hypercube and a cube connected cycles graph. The (perfect) hierarchical hypercube HHC_m can be constructed from a H_k with $k = 2^m$ by replacing each node by a hypercube H_m. It has $2^{2^m} \cdot 2^m = 2^{2^m+m}$ nodes.

Generalized Hypercubes and Hamming Graphs

The original hypercube is also called binary hypercube because the nodes are binary strings. In [12] and [111] the constant boolean radix of two is replaced by arbitrary radices, potentially different for each dimension. The resulting graph is called *generalized hypercube* (GHC) or Hamming graph. It will be regarded in Subsection 3.3.5.

2.6 Exemplary P2P Hypercube Gossiping Applications

Two utterly different applications demonstrate the universal appropriability of periodic peer-to-peer hypercube gossiping. They differ in the kind of utilized hardware, the type and complexity of the aggregation function, the disposition of samples, and the statefulness and significance of past samples.

2.6.1 Scalable Audio Communication for Massively Multiuser Virtual Environments

In *auditory virtual environments (AVEs)*, the participants' auditory perceptions correspond as closely as possible to their virtual surroundings [15]. With the advent of shared virtual environments came the desire to not only hear the background sounds of the environment but also talk naturally, that is, with an audio model conforming to the virtual environment. Just as in conferencing, each participant must be provided with a composition of the other participants' audio streams. Every participant's own samples are only played to the other participants and not to himself. Several audio streams are combined into one by digitally summing the corresponding audio samples from the incoming streams. At this, only the latest samples are of interest; older samples hold no value and, accordingly, nodes hold no state except for the samples currently being recorded or played back. However, whereas in classical audio conferencing only one or two people speak at a time and the conference size is limited to few tens of participants, in AVEs all participants are allowed to speak at once. Each one hears those who are in his virtual vicinity, with their audio volumes attenuated according to their virtual distance. As a consequence, participants can form conversing subgroups, while overhearing more distant conversations. With the rise of a new generation of massively multiplayer online games (*MMOGs*), massively multiuser virtual

environments (*MMVEs*) are now experiencing hundreds of thousands of concurrent users with an upward trend. Audiences and crowds of spectators at performances and sports events constitute extreme examples, whereat thousands of peoples engage in activities such as cheering, chanting and singing together. Hence the audio service must be designed as to cope with many simultaneous audio streams [126]. Whereas in the past MMOGs had distinct locations or rooms and each user was located in one or another, now MMVEs develop towards continuous spaces wherein users can move seamlessly.

The problem of providing audio communication to such environments can be regarded as a special case of sensor fusion, where the sensors are microphones, the streams are series of audio samples, and the aggregation function is the sum over all sources. Despite this seemingly straightforward reduction, the task poses several challenges:

Audio streaming is a real-time process. From the recording to the encoding, sending, receiving, decoding, aggregating to the playback, all operations need to complete within time bounds to avoid loss of streaming. On real-time operating systems, this is ensured by guaranteeing periodic CPU time slices to real-time processes. In a heterogeneous P2P environment, however, the employed hardware usually is of commodity class, the operating system generally does not support real-time scheduling and the application competes with other processes for CPU and network bandwidth.

For each listener the audio volumes of each speaker need to be attenuated according to the listener's distance to them. To this end, the virtual avatar locations must be transformed into node positions in the network. A clustering algorithm is responsible for mapping a given scene with the virtual locations of the participants onto a hypercube network topology, thereby assigning all participants their neighbors. At this, several criteria must be considered:

Scene Accuracy In particular, the algorithm should perform the clustering in an intuitive way, so that the participants' expectations (e. g. who will hear them) are approximated as closely as possible. For instance, two participants standing closest to each other should experience a low latency between them and hear each other loud and clear and with a high scene accuracy.

Bandwidth/Degree To keep network degree and bandwidth requirements as low as possible, the algorithm should minimize the number of hypercube dimensions.

Latency In voice communication, excessive latency affects conversation adversely in various ways. Hence, the solution must keep latency between conversation partners at a minimum.

Stability Furthermore, the algorithm should be stable to avoid drastic dynamic changes of cluster memberships. For instance, a single person walking around in the virtual environment should only have local effects and should not affect the whole network structure.

To summarize, audio communication for MMVEs comprises the provision of a large number of users with their personal, position dependent compositions of audio streams, in real-time, at low latency, using non-realtime components connected through a heterogeneous network with diversely limited bandwidths and varying delays.

Numerous architectures have been considered for audio delivery in conventional conferences as well as virtual environments. They differ greatly in their assumptions regarding network topology, features, and available bandwidths. Although some of the games provide audio systems already, these still have problems coping with dense crowds created by large numbers of users gathered at the same virtual location, e. g. attending the same events or

activities such as New Years Eve's countdown or popular sports events. In lack of functioning comprehensive solutions, unsatisfied users resort to detached and limited conferencing tools such as TeamSpeak, Mumble, Ventrilo, or Skype for "critical" communication.

In Chapter 8, a periodic peer-to-peer hypercube gossiping based approach towards scalable audio communication for MMVEs will be proposed. It comprises some considerations for the implementation as well as a prototypical mapping algorithm that clusters all participants standing in general position into a minimum diameter network graph.

2.6.2 Decentralized Data Fusion for Object Tracking

Decentralized data fusion (*DDF*) is an instance of the general distributed inference problem in which there is a single common state of interest—often a geographical area with an initially unknown number of physical objects (*targets*), their locations and velocities—that is observed by a number of distributed nodes, each equipped with one or more sensors, and connected by some (heterogeneous) network [106]. Each node periodically obtains observations from its sensors by which it estimates the state of interest. All nodes' estimates are merged (*fused*), along with previous estimates, to form a condensed estimate at each node that is more confident than one that any node could produce alone. Thus, in object tracking applications, past measurements matter: Each node holds a state comprising the objects' probability densities conditioned on all prior observations. In contrast to audio conferencing, every node's own measurements are included in its local aggregate. The employed hardware is likely dedicated and hosts a real-time operating system.

In recent years, the decentralization of data fusion techniques to combine data from multiple sensors has received increasing attention. It coincides with the emergence of a new driving paradigm of *Network Centrics*, e. g. Architectures or Operations (NCO) that scatter responsibility across the network. In 1994, guiding principles were postulated [57], amounting to the abolishment of central components such as single fusion centers, central nodes, common communication facilities, or total knowledge of network topology. The rationale behind it is to ensure scalability by absence of central bottlenecks as well as modularity and flexibility by obviating the need for knowledge of the network. Apparently however, adherence to these principles is necessary but not sufficient to ensure scalability.

Scaling DDF networks in a heterogeneous environment presents several challenges: Bandwidth limits the number of neighbors to which each node may send state estimates, whereas excessive latency vitiates the value of state estimates; particularly estimates of targets that behave in a non-linear way may decay quickly. As a consequence, latency must be kept to a minimum. In [30], the necessity to spread information "as fast and as efficient as possible" and the need for analytical performance models and metrics to predict latency and reliability is emphasized.

Another challenge generally encountered in DDF networks is the necessity to eliminate common past information resulting from the fusion of estimates that were descended from the same data. To avoid double-counting, this correlated information from past fusion events must be identified and removed, which requires the availability of the original data sets. Although the theoretic fundamentals have been well documented and studied for about twenty years now, the removal of common information while minimizing the amount of data exchanged still remains one of the main difficulties in distributed information fusion [29]. Numerous methods for DDF have been proposed, handling correlated data in diverse ways. Yet all in all, current approaches compensate this so-called *data incest* by approximations, have scalability issues with regard to latency or bandwidth, are fragile regarding single node failures, or resort to central components.

In Chapter 9, a periodic peer-to-peer hypercube gossiping based DDF approach will be proposed. The basic hierarchical structure of the hypercube permits efficient aggregation and dissemination of information with logarithmic time and bandwidth requirements while ruling out the possibility of double counting. In contrast to 1-trees, such as star and chain topologies, the dense structure of a hypercube sustains connectivity in the event of single node failures. The responsibilities of all nodes are fully determined by their position in the hypercube, thus obviating the need to negotiate communication schemes or carry information pedigree logs.

2.7 Problem Definition

The general task addressed by this thesis could be called *aggregation-gossiping* or *fusion-dissemination*. It is the combined problem of computing an aggregate from multiple distributed live measurements while distributing the result. The measurements are performed periodically and are repeated indefinitely. Each node needs, at all times, an aggregate that is fused from all nodes' as-current-as-possible information. Possibly, this task could be viewed as a specialization of the classic gossip problem, whereat each node's single piece of information is communicated to all other nodes; however, because of its distinctive periodic nature and the practical significance of in-network aggregation, it may well be considered a fundamental problem of its own. Undoubtedly, a solution should embrace the specific problem as well as potential optimizations.

Clearly, the solution should be scalable with regard to the number of nodes. Additionally, it should be robust and present no single point of failure. As a consequence, I regard a decentralized mode of operation: The task is to be performed symmetrically among equal peers with mostly identical qualities and responsibilities, particularly, without relying on any kind of distinguished node with special abilities.

Regarding the nodes' connectivity, I universally assume that every node is somehow connected to the Internet via a link of finite bandwidth. Since most of today's network interfaces are full-duplex, I assume that a node can send and receive data at the same time. All in all, this means that each node may communicate with any other, or several ones, but generally not with all of them during a given time interval.

In some parts of this work I furthermore assume that the sampling frequency exceeds the frequency with which datagrams can be sent to several neighbors. An obvious example of this is audio streaming.

Since real-time applications require a low latency, I look for solutions that exhibit a low average latency between measurement and consumption of information.

Finally, because live deployments of networks with non-trivial numbers of nodes are costly, I demand that the performance of the solution is assessable by means of both efficient and faithful simulation, or, even better, analytically.

The problem addressed by this thesis can thus be summarized as the problem of providing decentralized, bandwidth-efficient, low-latency fusion-dissemination of distributed (high-frequency) periodic measurements in heterogeneous environments, with predictable performance.

Chapter 3

Periodic Peer-to-Peer Hypercube Gossiping

Contents

This chapter presents *periodic peer-to-peer hypercube gossiping* as a means for efficient decentralized data fusion. It starts with the definition of terms, followed by the explication of the assumptions and requirements. Based on these, the network topology of the approach is established, and some notable properties are pointed out. Subsequently,

structural changes as a result of adding or removing nodes, deviations of the network from its fully allocated state, as well as a possible extension of the approach to generalized hypercubes are discussed. The subsequent section develops the communication and aggregation scheme whilst taking into account the periodic nature that is inherent to streaming communication. The section that follows gives an outlook on the composition of latency in a peer-to-peer context which will be examined in detail in Chapter 6. The chapter is concluded by a survey and classification of related work.

3.1 Nomenclature

m dimension of the hypercube topology

b base or radix of the hypercube, $b = 2$ for common (binary) hypercubes

d degree of a node, $d = (b-1)\,m$

N the number of nodes, $N = b^m$ for the fully allocated hypercube

L the number of levels or communication rounds equaling the dimension of the hypercube

Ł_ℓ level ℓ of the communication scheme, with $\ell \in [0..L-1]$

$\text{Ł}_{\ell_1} \cdots \text{Ł}_{\ell_2}$ set of levels $\{\text{Ł}_\ell \,|\, \ell \in [\ell_1..\ell_2]\}$,

$I(\text{Ł}_\ell)$ cumulative information received on Ł_ℓ.

$f_{\mathbf{samp}}$ sampling rate or sampling frequency of the nodes' sensors

g generator or initializer (see Section 2.3)

f fusion or merging function (see Section 2.3)

e evaluator (see Section 2.3)

T send period, i. e. time between consecutive packet transmissions

$c_b(m,h)$ number of nodes being h nodes away from an arbitrary node n, of a b-ary hypercube of dimension m

χ_{n_1,n_2} probability distribution of the delay it takes to send a datagram from node n_1 across the underlay network to node n_2

$\chi_{n_1,n_2,q}$ q-quantile of χ_{n_1,n_2}, i. e. delay that will, with probability q, not be exceeded

$\text{nb}_b(n,\ell,i)$ i^{th} neighbor of node n on Ł_ℓ on a b-ary hypercube

$\text{nb}(n,\ell) := \text{nb}_2(n,\ell,0)$, the (sole) neighbor of node n on Ł_ℓ on a binary hypercube

$\hat{u}$ summit of node u, denotes a node's largest complete containing hypercube (see Subsection 3.4.2)

$\overline{u}$ maxlevel of node u, denotes the highest level ℓ for that $\text{nb}(u,\ell)$ exists (see Subsection 3.4.2)

3.2 Assumptions and Requirements

3.2.1 Assumptions

The approach presented in this dissertation rests on the following assumptions:

- Sensor data becomes available periodically at each node n, at times $t_n + i/f_{\text{samp}}$, $i \in \mathbb{N}_0$.
- Each node is connected to the Internet through some sort of Internet access link. Every node's access link may have different latencies and bandwidths associated that may also differ between upstream and downstream direction.
- The transmission of data incurs an overhead. As a consequence, each node can communicate only with a limited number of nodes per time interval.
- Some nodes may be located behind NAT routers or firewalls rejecting unsolicited messages from the outside, hold no public IP address of their own, or may not even know the outgoing public IP address of their router.
- Backbone network delay may vary for different sender/receiver pairs (u, v) (*locality*) and may even be different for each consecutive message (*jitter*). Particularly, messages may arrive out of order. Moreover, messages may not arrive at all (*packet loss*). Apart from that, the interconnection network (Internet) is assumed to have unlimited capacity, i. e. there is no limit to the global number of transmissions per time interval. Also, all backbone communication links are deemed independent, i. e. for any four distinct nodes u, v, w, and x, communications between u and v have no noticeable effect on communications between w and x.
- Every node has a clock, but the clocks are not guaranteed to be synchronized. There is no universal synchronization signal.
- Nodes may not be able to hold, process, and/or communicate the measurements, underlay addresses, or possibly not even the overlay IDs of the entirety of nodes.
- The local sensor and the network link are the only sources of information available to each node, particularly, no node has total knowledge about the system.
- Nodes may deteriorate or fail at any time without prior notice.
- There are no malicious nodes.

3.2.2 Requirements

- The system shall provide decentralized bandwidth-efficient low-latency gossiping of high-frequency periodic measurements in heterogeneous environments.
- Wait latency (see Subsection 3.5.2 on page 49) shall be reduced as far as reasonable and practicable.
- The network shall not be limited to distinct sizes such as powers of two.
- The network configuration (*bootstrapping*) must be accomplished without the coordination of a central authority.

- A newly arrived node must be able to join the network given only the underlay address of an arbitrary contact node that is already part of network.
- The number and volume of the messages required to integrate the new node shall be well below $\mathcal{O}(N)$. The additional work load exerted on nodes contriving the accommodation shall be small in relation to their aggregation workload.
- The network shall be able to cope with a high node arrival rate, e. g. one that is proportional to the size of the network.
- The placement scheme must respect available bandwidth and shall be able to accommodate a small portion of bandwidth impaired nodes.

3.3 Network Topology

3.3.1 Finding the Right One

In Subsection 2.1.3 the *gossip problem* was presented as the task of finding a communication strategy such that every node learns all other nodes' information. For real-time stream gossiping, the challenge lies in finding a feasible network topology and communication strategy such that, in the face of limited bandwidth per node, each node learns from every nodes' measurement in as little time as possible.

Let us first assume an equal finite bandwidth for all nodes. For a time interval of any finite size, each node can communicate with a finite number of nodes. In a typical peer-to-peer context, with full-duplex Internet access paths, receiving and sending can happen at the same time.

We let t denote the smallest time such that, for an arbitrary number $x \in \mathbb{N}$, a node can both receive from x nodes and send to x nodes in the time $x \cdot t$. t then corresponds to the time in which one round of a two-way communication algorithm (2.1.4) can be executed. In full-duplex scenarios, t is determined by the lesser of the upstream and the downstream bandwidths, which for consumer links usually is the upstream one.

Our goal is thus to find a network topology and communication strategy for that the gossip problem can be solved for a high number of nodes in a small number of communication rounds in two-way mode: a network topology and communication strategy with a low gossip complexity.

Comparison of Gossip Graphs

In [65], several topologies are surveyed with regard to their two-way gossip complexity. Table 3.3.1 shows gossip times for common networks in two-way mode. In Figure 3.3.1 the lower and upper bounds on gossip time of these topologies are plotted for networks of different sizes (log). Several things can be observed at once:

- The Path and Cycle topologies have linear effort and are obviously poor choices for scalable gossiping.
- The Hypercube and its variations Cube Connected Cycles CCC_k, Shuffle-Exchange SE_k, Butterfly BF_k, and DeBruijn network DB_k, form a family of networks whose performance differs roughly by constant factors.
- Only the hypercube matches the (optimal) performance of a complete graph. Adding further edges to a hypercube graph will not improve gossiping performance.

Table 3.3.1: Gossip times for common networks in two-way mode [65]

Graph	Nodes	Diameter	Lower Bound	Upper Bound
Complete K_n	n	1	$\lceil \log_2 n \rceil + \text{odd}(n)$	$\lceil \log_2 n \rceil + \text{odd}(n)$
Hypercube H_k	2^k	k	k	k
Path P_n	n	$n-1$	$n - \text{even}(n)$	$n - \text{even}(n)$
Cycle C_n	n	$\lfloor n/2 \rfloor$	$\lceil n/2 \rceil + \text{odd}(n)$	$\lceil n/2 \rceil + \text{odd}(n)$
Cube Connected Cycles CCC_k	$k \cdot 2^k$	$\lfloor 5k/2 \rfloor - 2$	$\lceil 5k/2 \rceil - 2$	$\begin{cases} \lceil 5k/2 \rceil - 2 & k \text{ even} \\ \lceil 5k/2 \rceil + 1 & k \text{ odd} \end{cases}$
Shuffle-Exchange SE_k	2^k	$2k-1$	$2k-1$	$2k+5$
Butterfly BF_k	$k \cdot 2^k$	$\lfloor 3k/2 \rfloor - 2$	$1.9770k$	$2.25k + o(k)$ [a]
DeBruijn DB_k	2^k	k	$1.5965k$	$2k+5$

[a] $f(N) = o(g(N))$ denotes the fact that for an arbitrary number $c > 0$ a value N_0 exists such that $\forall N > N_0 : f(N) < c \cdot g(N)$ [92].

- The hypercube's relatives not only perform worse but also have distinct upper and lower bounds. This hints at the complexity of finding or proving an optimal algorithm for these topologies.
- The hypercube related graphs and communication schemes are defined only for distinct network sizes that are powers of two.

The following subsection will examine the hypercube as the "prime candidate with issues" in greater detail.

3.3.2 Presenting the Hypercube

By using a hypercube topology, the gossip problem can be solved optimally, i. e. in $\log_2 N$ communication rounds with regard to the total number of nodes N. Furthermore the hypercube-based solution is simple and straight-forward as we will see in Section 3.4.

As particularized in Section 2.5, a (binary) hypercube of dimension m, denoted H_m, is defined as the graph whose nodes are all binary strings of length m and whose edges connect those strings that differ in exactly one place.

H_m has 2^m nodes, $m \cdot 2^{m-1}$ edges and diameter m. Each node has exactly degree m [66]. Figure 2.5.1 on page 16 shows the network topology and node numbering for a binary hypercube network with $m = 3$, also called *3-cube*. The edges of a hypercube can be partitioned in a natural way according to the dimension they traverse. An edge is called edge of dimension k if it connects two nodes differing in the k^{th} bit [92].

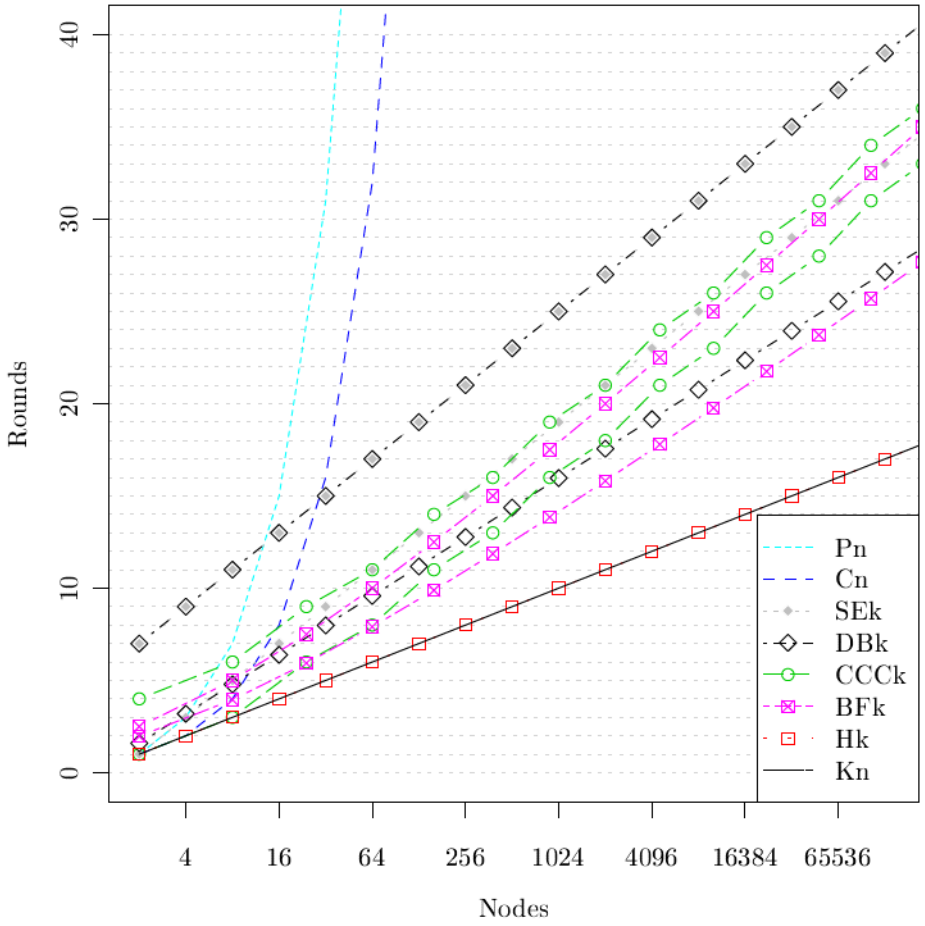


Figure 3.3.1: Lower and upper bounds on gossip time for common networks in two-way mode

In a network with N nodes each node n is connected to m neighbor nodes, i. e. one per dimension. The node ID of the ℓ^{th} neighbor can be obtained by simply flipping the ℓ^{th} bit of n's node ID:

$$\text{nb}\,(n, l) = n \,\text{xor}\, 2^{\ell}$$

where **xor** denotes the bitwise exclusive disjunction.

Robustness and symmetry properties are stated in Subsection 2.5.2.

Incomplete Allocations

The hypercube is defined only for sizes that are a power of two. Obviously, for a peer-to-peer network, a solution is required that works for any number of nodes. In [77] this gap was bridged by the proposed construction of so called *incomplete hypercubes* from an arbitrary number of nodes, which will be used henceforth.

An incomplete allocation of N nodes can be viewed as a descending series of complete hypercubes with dimensions corresponding to the positions of set bits in the binary representation of N. For example, a network of eleven nodes can be constructed by connecting

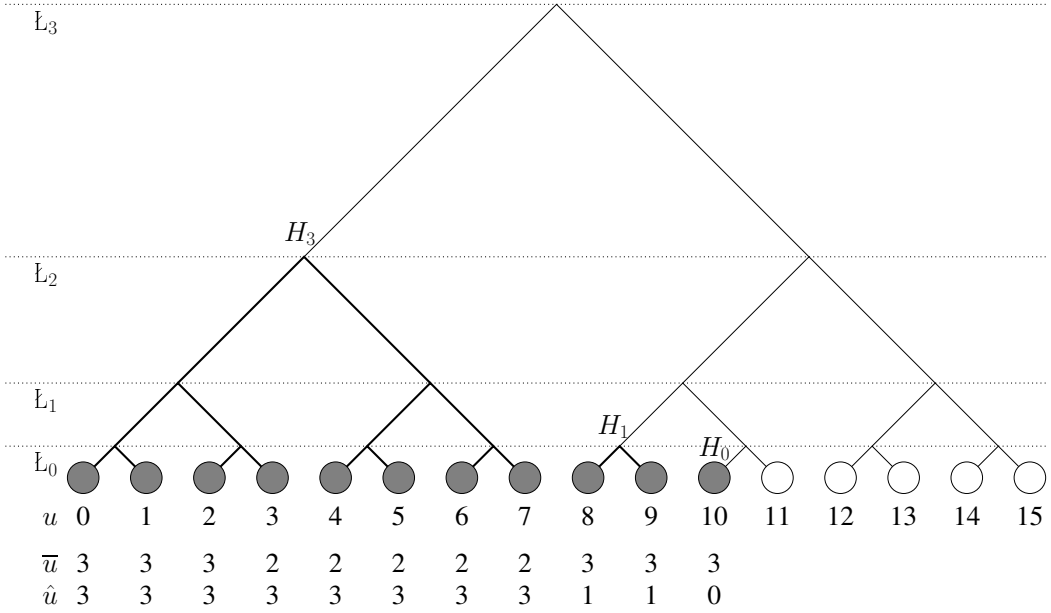


Figure 3.3.2: Incomplete allocation: Network of eleven nodes

a hypercube of eight nodes (H_3) with one of two nodes (H_1) and the zero-dimensional hypercube (H_0) of one node, as shown in Figure 3.3.2. Alternatively, the incomplete allocation can be thought of as the next higher complete allocation with the ID-wise topmost nodes and their connections removed. The corresponding network graph is shown in Figure 3.3.3. For some theoretical considerations that follow, we will nevertheless only regard

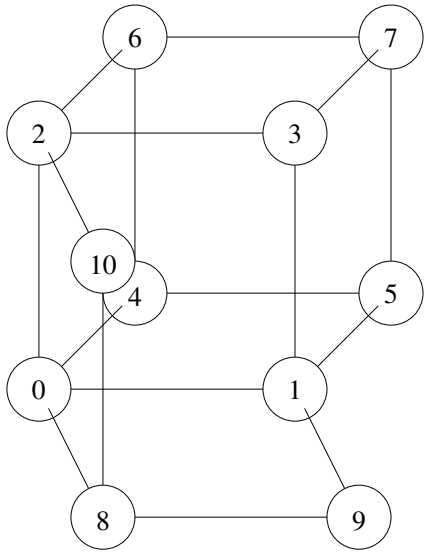


Figure 3.3.3: Network graph for eleven nodes

the fully allocated hypercube so as not to unnecessarily complicate the formulas. As a matter of course, in the implementation presented in Chapter 7, all incomplete allocations are accounted for.

Overlay Distance

The *overlay distance* h between two nodes denotes the number of overlay hops a packet travels from one node to the other. The *network diameter* is defined as the maximum value of h over all node pairs, i. e. the greatest distance between any two nodes.

In Subsection 2.1.3, *Gossip complexity* was introduced as the necessary and sufficient number of rounds to complete the gossiping problem. Obviously, the diameter is a lower bound of the gossip complexity; the gossiping scheme must have at least this many communication rounds. For the hypercube topology, it is also the upper bound. That is, the longest overlay path, called *maximum path*, is determined by the gossip complexity and amounts to exactly m hops.

For deriving traversal time and wait latency estimates, it is obligatory to not only consider maximum paths but also to investigate the distribution of hop distance. We define $c_b(m,h)$ as the number of nodes being h nodes away from an arbitrary node n in a binary hypercube with dimension m:

Theorem 3.3.1.

$$c_2(m,h) = \begin{pmatrix} m \\ h \end{pmatrix} \text{ with } h \in [1,m] \tag{3.3.1}$$

Proof. To reach a position that has a (Hamming) distance of h, h dimensions must be traversed. Since h dimensions are chosen from m possible ones, the number of combinations is given by the binomial coefficient. □

E. g. the number of nodes that can be reached within one hop equals the network degree:

$$c_b(m,1) = m$$

Table 3.3.2: Overlay hop count distribution $c_2(m,h) = \binom{m}{h}$

$c_2(m,h)$ $m\downarrow, h\rightarrow$	0	1	2	3	4	5	6	7	8
1	1	1	0	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0	0
3	1	3	3	1	0	0	0	0	0
4	1	4	6	4	1	0	0	0	0
5	1	5	10	10	5	1	0	0	0
6	1	6	15	20	15	6	1	0	0
7	1	7	21	35	35	21	7	1	0
8	1	8	28	56	70	56	28	8	1

Table 3.3.2 shows the hop distance distribution for binary hypercubes H_1 to H_8. The values in the columns equal those of the diagonals of Pascal's triangle. From each row, the cardinality of nodes at a certain distance can be read. The fourth row, for instance, shows the distribution for a network of size $m = 4$, i. e. 16 nodes: Each node has 4, 6, 4, and 1 peers at a hop distance of 1, 2, 3, 4, respectively. As can be observed on the diagonal of the table, the maximum path length m occurs for only one destination per source (equivalent to one source per destination).

3.3.3 Deformation

In some cases the perfect hypercube topology is in fact undesirable. One such case when to deviate from this topology is when accommodating bandwidth impaired nodes. The perfect hypercube is fully symmetric. For every node, the bandwidth required in each direction grows in the order of $\log N$. While this is a moderate requirement in general, it may be too much for some nodes[1]. This minority of nodes with insufficient bandwidth can be accommodated by relocating them up in ID space, into the sparser regions of the incomplete hypercube, described above, where they will have fewer neighbors. This is shown in Figure 3.3.4 by example of node 111. This scheme allows to flexibly scale down

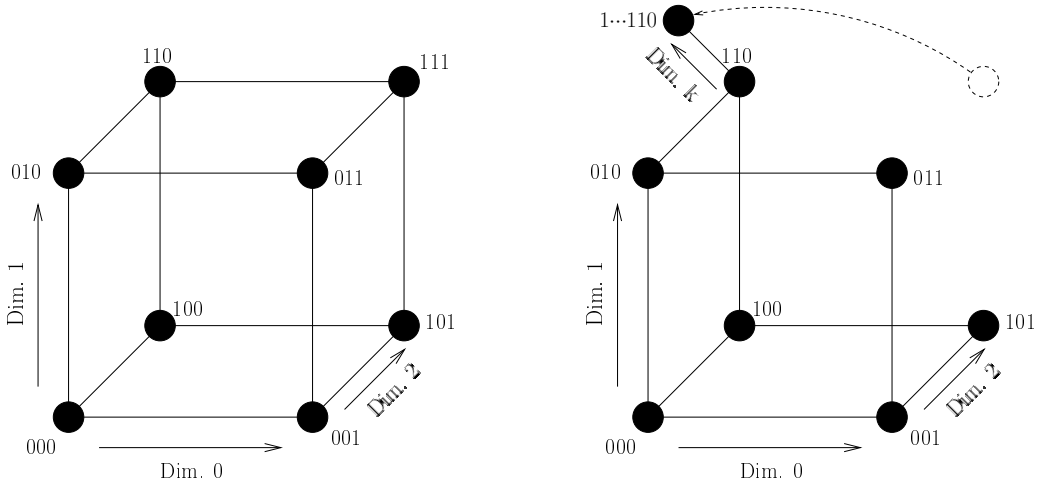


Figure 3.3.4: Deformation: Bandwidth impaired node 111 is relocated into sparse region where it has only one neighbor.

single nodes to just one connection per direction without inflicting additional latency on the rest of the network.

Insufficient bandwidth in general would have to be handled differently, e. g. by application specific solutions shown in Chapters 8 and 9 or by resorting to a constant degree network, such as the butterfly or cube connected cycles topologies, albeit at the cost of longer paths, consequently resulting in an increased latency.

3.3.4 Dynamics

Until now we have only regarded properties of an already configured, static network. In a peer-to-peer context, however, not only the initial construction (*bootstrapping*) of an overlay network, unaided by any central authority, must be considered but also the continual dynamic addition and removal of nodes. Below, we will regard such changes to the network topology, henceforth subsumed as *node dynamics*.

[1] Figure 4.5.2 on page 68 shows the distribution of upstream bandwidth, measured worldwide in March, 2010. About 15 % of all hosts in the data set exhibit an upstream bandwidth of less than 200 kbps

Node Arrival

Arriving nodes must be assigned an overlay ID and be introduced to their prospective neighbors. In order to ensure scalability, this must be accomplished without the coordination of a central authority. An arrival shall be able to contact any node that is already part of the network. As stated in Subsection 3.2.2, the number and volume of the messages required to integrate the new node shall be well below $\mathcal{O}(N)$. The network shall be able to cope with a high arrival rate. The placement of the nodes must respect the available bandwidth and shall lead to a highly efficient overall topology. Also, in peer-to-peer contexts, a considerable portion of the nodes may be behind NAT routers and hold no public IP address of their own. The conjunction of these constraints makes network bootstrapping and node accommodation no trivial task. Fortunately, with a hypercube gossiping network, we have a system at hand that allows for efficient aggregation. By leveraging the network's powerful gossiping and aggregation facilities we can actually reach a solution that satisfies all of these requirements.

Since gossiping allows us to disseminate all management information to all nodes, we can design a network that can be joined from an arbitrary contact node. For small networks we can disseminate the whole network topology. In fact, this information is contained in every final aggregate at all nodes if the sensor data is tagged with the node's ID and these tag sets are fused along with the data. With the complete topology at hand, the perfect location for a new arrival can be determined by every node and the placement can be communicated to the whole network.

For large networks, however, we may not assume that the whole network structure can be disseminated as this would likely result in traffic and memory requirements in the order of $\mathcal{O}(N)$. Thus, joining the network must not require knowledge of the network as a whole.

To join the network, an arrival contacts a random network node, called the *contact node*, by sending a message to its socket address[2]. Upon reception of the message, the contact node tries to accommodate the arrival by first determining, with the help of the network, a vacant overlay ID, called *join address*, and a *host,* that will become the lowest level neighbor of the new node. Depending on the available bandwidth of the joining node, the placement strategy differs, as expatiated below.

Placement of Regular Nodes To attain a perfect hypercube, the join address should be chosen as the minimum node ID that is not already part of the network. A naïve solution might just determine and disseminate this ID by the aggregation network and simply assign this ID to every arrival. However, for high arrival rates, this will cause congestion near that node when many nodes try to join the network at the same time. Also, growth would be limited by the re-computation of the new minimum.

The proposed solution borrows from the natural phenomenon of crystallization, a self-organizing process resulting in the formation of solid crystals precipitating from a solution, melt or gas. At this, molecules of the solute attach to the crystal in a stochastic but organized fashion. Information about the entirety of the crystal is nowhere required. Molecules can attach to the crystal simultaneously and independently. Yet the result exhibits a highly regular structure. Whereas the creation of a perfect crystal may take a very long time, the growth process can be accelerated by further supersaturating the solute. This results in fast growth at the hazard of some imperfections.

[2]The contact node's socket address may be obtained, e. g. from a list of past addresses, some service or by sending packets to a well known port at random IP addresses.

Similar principles can be applied to guide the growth of the hypercube network. A relaxation of the node placement discipline facilitates a faster growth but leads to bloat, resulting in a topology that is not as compact as it could be. Hence, the proposed strategy is a dynamic one, subject to the arrival rate, which is easily observable for every node: For low arrival rates, the network growth should result in a perfect hypercube, for high arrival rates the algorithm should allow for rapid growth at the expense of some regularity. Instead of using only the minimum node ID as a single docking location for arrivals, sets of feasible locations, referred to as *holes*, are collected all over the network. A subset thereof is passed to neighbors. The target location is chosen randomly from the set union of local and remote options. At this, closer target locations are preferred. The selection of the subset acts as a regulating screw that is in turn adjusted as a function of the current node arrival rate. For very low arrival rates (less than one arrival per d rounds), the subset is chosen as the minimum vacant node ID. This leads to a perfectly compact hypercube with minimum communication overhead. For medium rates (k up to k_{max} arrivals per d

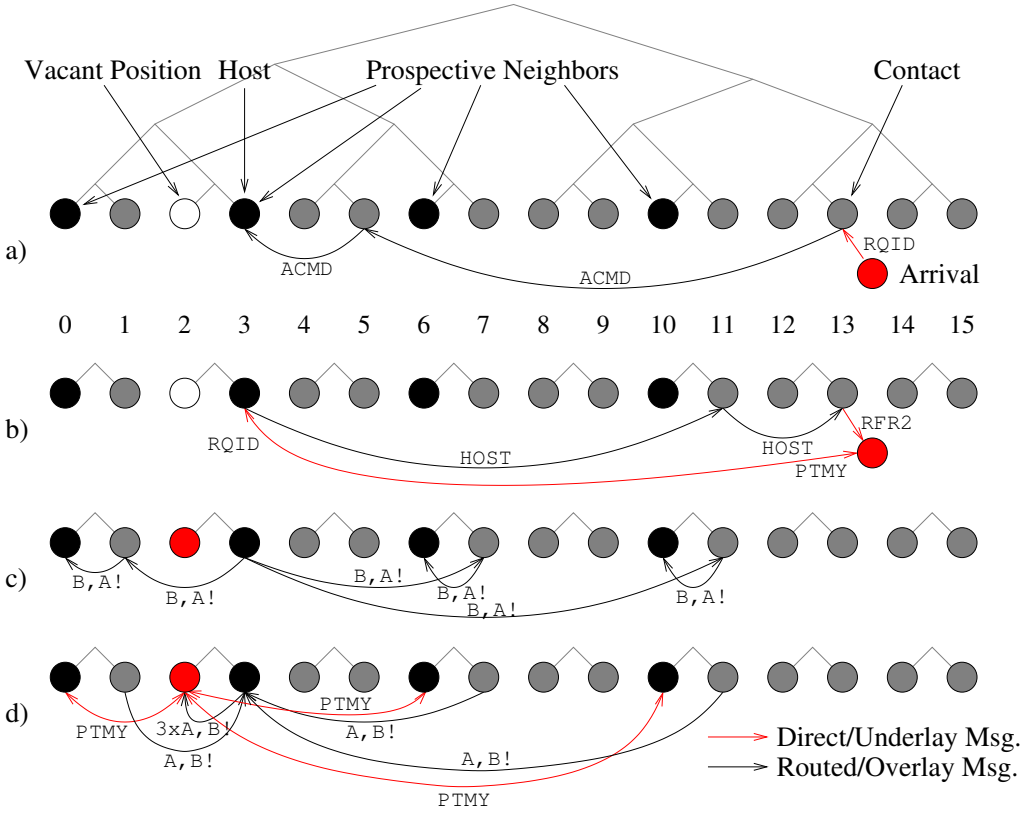


Figure 3.3.5: A new node gets to know its neighbors. a) Arrival requests ID from contact. Contact routes accommodation request towards 3, being the lowest level neighbor of 2. So does 5. b) 3 accepts host status and routes acknowledgment to contact. Contact refers arrival to host. Arrival requests ID from host. Host welcomes arrival. c) Host introduces arrival to arrival's neighbors. d) Introductions are returned to arrival. Arrival and its neighbors welcome each other. All black messages are routed piggy-back with application payload.

rounds), the bottom-k holes are propagated, resulting still in perfectly compact hypercubes at a k times higher communication overhead. For still higher rates, instead of bottom-k_{max}, k_{max} *random* holes having IDs smaller than a threshold consisting of the total number of nodes multiplied by a constant factor w are chosen. Note that since the resulting sets of holes are different for each contact, the number of nodes that can join within d rounds is not limited by k_{max} but by the threshold. The communication overhead is bounded by k_{max} and the bloat is bounded by w. As soon as the growth rate is decreasing to medium rate, new arrivals fill the accrued holes and the network is becoming compact again. k_{max} should be chosen with respect to application specific payload size, so that the management overhead of communicating the holes is small in relation to the total packet. The choice of w determines the bandwidth efficiency of the network during continuing high growth rates. A sensible choice is $1 < w < 2$.

After having determined a candidate host, the contact node routes the accommodation request in the direction of that host, i. e. a neighbor node of the current node is chosen whose node ID has a smaller Hamming distance to the prospective host than the current node's ID. At the neighbor the request is re-evaluated. Since a host with minimal distance was chosen, and that distance has now been decreased by one hop, the neighbor is likely to choose the same target host or one that is in equal distance. As as consequence, the distance to the target node decreases with each hop the request travels as long as the information is consistent.

Figure 3.3.5 shows the arrival of a new node (red) at node 13 of the hypercube network. The smallest ID not present in the latest aggregate of node 13 is 2, and is given to the arrival as its prospective address (*destination*). The existing neighbors of node 2 are 3, 0, 6 and 10, the lowest level neighbor is 3.

Placement of Bandwidth Impaired Nodes Nodes with insufficient bandwidth for d full-duplex connections cannot be integrated into the dense portion of the ID space. Instead, they need to be placed in the sparse regions and linked via higher dimension connections to hosts in the dense region, as described in Subsection 3.3.3. These additional high-dimension links should be spread evenly among the nodes in the dense region. For this, we cannot rely on the random allotment of contact nodes, as the choice of contact nodes may be biased. One obvious reason for this is that contact nodes cannot be firewalled. Also, if contact addresses are retrieved from a service, this service may know only a small portion of all nodes. On that account, a random host needs to be determined explicitly. But also the direct computation of a random host ID is problematic since the contact node generally cannot know whether the target host exists. Rather, the proposed solution is to route the accommodation request along a random walk. By constraining the routing steps to those that increase the Hamming distance to the contact node, cycles can be avoided without the need to record the route. At every node, the walk may end. The alternative is to continue the walk by traversing one of the feasible edges, that increase the distance to the contact node. Clearly, if all hosts are to be favored equally, the probability to stop the walk needs to increase with the distance to the contact node. After having taken d steps, each feasible edge is chosen with the probability $t(d)$; we let $s(d)$ denote the (remaining) probability with which to stop there. The probabilities of all options obviously must sum to one, so at every node the following applies:

$$s(d) + \sum t(d) = 1.$$

In Figure 3.3.6, the stop and traverse probabilities for a random walk on H_3 are illustrated. The walk starts at node 111. With a probability of $1/8$ the walk ends right at the start. This

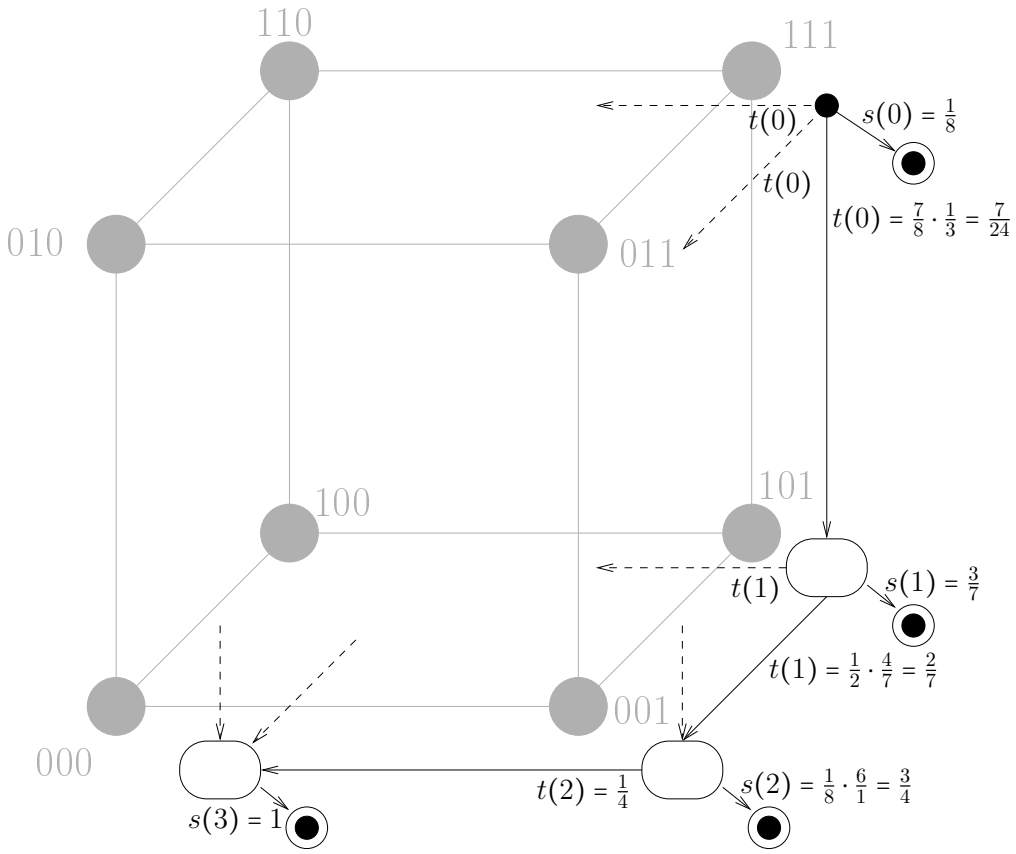


Figure 3.3.6: Stop and traverse probabilities for random walk on H_3

is intuitive if one considers that each node should be reached with exactly this probability. The outgoing edges are chosen with $\frac{7}{8} \cdot \frac{1}{3} = \frac{7}{24}$. The probability to end our path at the next node is determined by 1/8 divided by the probability for having reached this node. When the walk has reached the most distant node there are no feasible paths left and the probability to end the walk equals 1.

Theorem 3.3.2.

$$s(d) = \frac{1}{N \cdot d! \cdot \prod_{h=0}^{d-1} t(h)} \tag{3.3.2}$$

$$t(d) = \frac{1 - s(d)}{m - d} \tag{3.3.3}$$

Proof. The product represents the probability of a path leading to the current node, and $d!$ accounts for multiple paths leading to the same current node. $m - d$ equals the number of outbound edges at the current node. □

Corollary 3.3.3. *Substituting* (3.3.3) *into* (3.3.2) *gives*

$$s(d) = \frac{1}{N \cdot d! \prod_{h=0}^{d-1} \frac{1-s(h)}{m-h}} = \frac{1}{N \cdot d! \frac{m-d!}{m!} \prod_{h=0}^{d-1} (1-s(h))} = \frac{\binom{m}{d}}{N \prod_{h=0}^{d-1} (1-s(h))}$$

Introduction and Hole Punching As particularized in Subsection 2.4.5, firewalled nodes cannot start communicating with each other without having been introduced to each other first. A connection between two firewalled nodes needs to be initiated from *both* sides. This must be regarded in all initial communication. On this account, once a host has been found, word is sent on two different paths back to the applicant: directly by the host, using the applicant's underlay address, and by routing the message through the contact node. Likewise, the new node is introduced to all higher level neighbors by the host.

Neighbor Discovery If the introduction fails for any reason, nodes may not know all of their neighbors. They will then advertise their missing neighbors as holes, as described in Paragraph "Placement of Regular Nodes" above. A node receiving a set of holes can use this information in several ways: If it finds itself or any of its neighbors in the set of holes, it informs the respective host and neighbor. Likewise, if it finds a host that happens to be one of its own missing neighbors. This way, missing edges are completed.

Conflict Resolution If both introduction and neighbor discovery have failed, it may happen that missing neighbor locations are filled up by new arrivals. This leads to a situation where one node ID is occupied by multiple nodes. This conflict is likely discovered during the following introductions when two nodes claim to be some other node's neighbors, if not, during the exchange of the hole sets. When any node learns of two different nodes (underlay addresses) that claim to be associated with the same overlay ID, it introduces them to each other. The conflicting nodes then engage in a battle for the ID. In the current implementation, premising the absence of malicious nodes, the battle is won by whichever node claims to be part of the greater network. The loser forfeits the ID and requests a new ID from the winner. Additionally, both nodes' neighbors are introduced to each other. Thereby conflict resolution is propagated and the lesser of the two conflicting hypercubes shatters on the greater one, the shards further adding to the greater one's size.

Node Failure

Well-behaved nodes will send a termination message before going offline so that neighbors may immediately reconfigure their connections. Even so, robustness requirements of decentralized fusion networks demand that systems remain functional in the event of unforeseen node failures. For singly-connected or 1-tree networks, such as star and chain topologies, the failure of any non-leaf node will separate the network in two components [143]. In denser networks without redundant paths, like the proposed hypercube topology, node failures will result in the non-appearance of information sources at other nodes but not break the network.

As will be shown in Theorem 3.4.11, an instantaneous node failure in a previously complete hypercube network will, until corrective action is taken, result in the loss of some streaming sources. This will reduce the accuracy of the result of some nodes until reconfiguration, but no node will become isolated. When not receiving any messages from a neighbor for some time, a node assumes its neighbor gone, reconfigures its connections, and

starts advertising the vacant position, just like it does after having received a termination message.

If ping time for just one neighbor keeps increasing, this indicates congestion on its link. The node will notify the neighbor of this fact. If the neighbor receives several congestion messages or notices increasing ping times for the majority of its neighbors, it will terminate its connections and request a bandwidth reduced position.

3.3.5 Generalization

The original hypercube, also called *binary hypercube*, is defined through the notion of binary strings. In [12] and [111] the constant boolean radix of two is generalized to arbitrary radices. The resulting graph is called *generalized hypercube* (GHC) or Hamming graph. This generalization is also applicable to the gossiping network. For real-time gossiping applications, the choice of radix is a trade-off between bandwidth and latency. Abundant bandwidth provided, higher radices will, for a given number of nodes, result in fewer dimensions m and communication rounds L. Depending on the composition of network delay[3], this can reduce latency. Figure 3.3.7 shows a three-dimensional generalized hypercube of

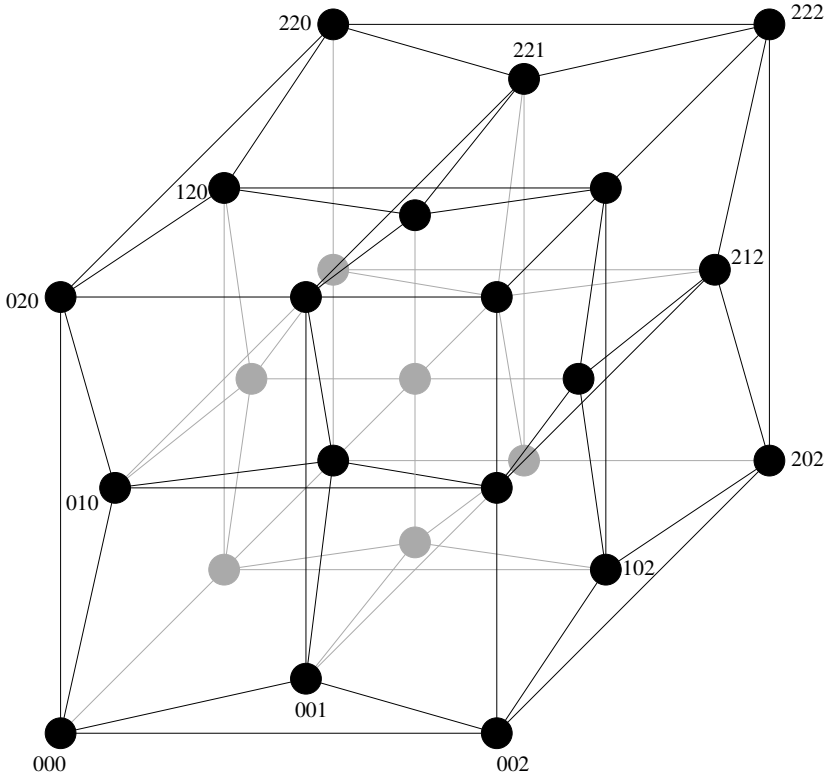


Figure 3.3.7: Generalized hypercube $b = 3, m = 3$, node IDs in ternary notation

radix three. Whereas in H_3 each node has just one neighbor per dimension, here each node has two. Specifically, each node has $b-1$ neighbors on each of m dimensions, resulting in a degree of $d = (b-1) \cdot m$. Two nodes, numbered i and j, with $i, j \in [0, N)$, are now

[3] particularly for high bandwidths and high propagation delays

neighbors on dimension k iff their base-b representations differ exactly in the k^{th} digit, or alternatively, iff

$$|j-i| \bmod b^{\ell} = 0 \ \wedge \ \lfloor \frac{i}{b^{\ell+1}} \rfloor = \lfloor \frac{j}{b^{\ell+1}} \rfloor.$$

Accordingly, for generalized hypercubes, the overlay distance, introduced in Subsection 3.3.2, becomes

$$c_b(m,h) = (b-1)^h \left(\begin{array}{c} m \\ h \end{array} \right) \quad \text{with } h \in [1..m]$$

because, again, h hop-levels are chosen from m possible ones and each hop is now to one of $b-1$ neighbors.

For generalized hypercubes we regard the communication problem as an instance of the F-$(b-1)$ gossip problem (see Section 2.1), where in every communication round each node communicates with $b-1$ neighbors, i. e. messages to several neighbors on the same dimension are sent within the same logical round of the communication scheme, as explained in the next section.

Given a certain bandwidth limit, resulting in a maximum degree, several network configurations with regard to the radix b are possible. Figure 3.3.8 shows the cumulative hop-count distribution for all possible configurations having a degree less than or equal to 5. The height of a (sub-)column represents the number of nodes from which a node has a distance corresponding to its color, e. g. the degree $d = c_b(m,1)$ can be read from the light bars at the bottom. As expected, the graph shows that for small networks ($N \leq 6$), by

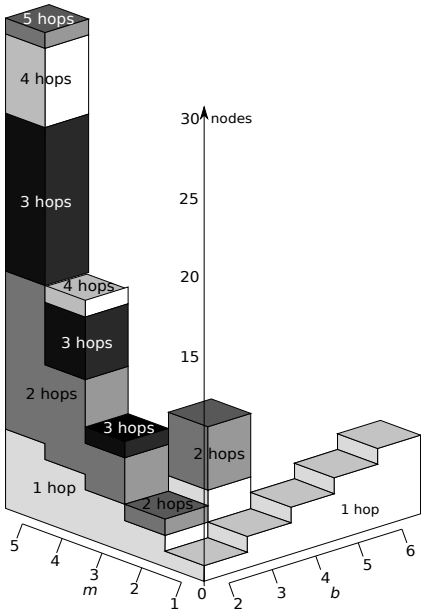


Figure 3.3.8: Hop-count distance distribution for feasible generalized hypercube network configurations with a maximum node degree of $d = 5$. The number of peers that are 1 hop away from any node is indicated by the light bar at the bottom, peers that are 5 hops away medium gray at the very top.

setting $m = 1$, all nodes can communicate directly with each other, i. e. over a distance of just one hop. In this case the network topology is that of a *complete graph*. For network

sizes of 7 to 9 nodes, the configuration $b = 3$, $m = 2$ yields the lowest hop-count. For larger networks the most bandwidth efficient configurations with $b = 2$, $m \geq 4$ need to be used. Note that in this case the maximum hop-count occurs for just one peer of each node.

3.4 Communication and Aggregation Scheme

The network should support various kinds of aggregation, from simple count and maximum queries to audio conferencing to the aggregation of multidimensional probability densities. In order to abstractly formulate aggregation, we follow the concept described in [98] and recapitulated in Section 2.3. We define three functions:

- An initializer or generator g,
- a merging or fusion function f, and
- an evaluator e.

The total aggregate is computed by the function composition of e, f, and g:

$$agg = e\Big(f\Big(\cdots f\big(f\big(g(),g()\big),f\big(g(),g()\big)\big)\cdots\Big)\Big)$$

The generator g takes raw sensor data and prepares it to be fused by function f. The recursive invocation of f, also denoted as functional power f^k, accomplishes the merging of information. Although the result of the last invocation of f already contains all required information, it may not be the answer to the query. Therefore a terminal invocation of e is needed to produce the final aggregation result. Where f is commutative and associative, the fusion of a set $S = \{x_0, \ldots, x_n\}$ of pieces of information is defined as

$$f(S) := f\Big(f\Big(\cdots f\big(f\big(f(x_0,x_1),x_2\big),x_3\big)\cdots\Big),x_n\Big).$$

Tags Optionally, the generator g may tag the sensor data with metadata, e. g. the ID of the originating node, a timestamp, information on the measurement or sensor, or even a digital signature. The fusion function f would then tag its result with the set union of the tags of its arguments. The use of tags serves several purposes: For one thing, it gives information on the significance of the final result, e. g. a result obtained from the fusion of 300 sources may be more meaningful than one from 3. Also, the tags may contain signatures, by which the authorities of the individual information sources can be asserted. In [28] various benefits of tagging sensor data with metadata are particularized.

Apart from application specific benefits, the metadata can provide an up-to-date status on the aggregation network itself, that may be used for network management, e. g. to initiate structural repairs or determine target locations for joining nodes. For this, the node ID of each contributing source is sufficient. Due to the regular fusion scheme of the hypercube, such tag sets can be encoded very efficiently[4]. Still, for very large networks, the communication of tags may be infeasible. It is important to note that whereas the network can make use of such metadata for optimizations, the proposed approach does not depend on this information for network management.

[4] As a simple coding consider converting the ID set into a bit set representation and then encoding run lengths of zeros and ones using, e. g. Elias coding [48]. This could be further improved by taking into account the fusion scheme and resulting probabilities for possible resulting ID sets.

Conventional Communication Scheme In Subsection 2.1.4, the pairwise two-way communication scheme was defined as the *sequence* of sets (matchings) $E_i \subseteq E$, where for each $i \in \{1, \ldots, r\}$, all (x_1, y_1), $(x_2, y_2) \in E_i$

$$\{x_1, y_1\} \neq \{x_2, y_2\} \text{ implies } \{x_1, y_1\} \cap \{x_2, y_2\} = \varnothing.$$

The declarative communication scheme for one-shot gossiping on the hypercube was shown in Subsection 2.5.3. It is the solution to the problem of gossiping one piece of information per node to all other nodes. Performance is measured in rounds to complete the scheme. By contrast, the problem addressed by this dissertation is that of indefinitely gossiping a *stream* of information per node to all other nodes. Performance is measured by the currentness of information upon its arrival. Obviously, prerequisites as well as requirements are utterly different. So, before defining the communication scheme for periodic hypercube gossiping, we must first take a closer look at the characteristic periodicity that is inherent to all streaming communication.

3.4.1 Periodicity

In customary streaming applications, all communication takes place repeatedly, with a constant time interval that we refer to as *period*, denoted $T = const$. If, for instance, at time t_x node 5 sends data to node 7, it is assumed to also do so at all times $t_x + kT$, $k \in \mathbb{Z}$. The data may be different every time, the transmission event is not. If node 7 receives the data at time t_y, it also does so at $t_y + kT$. The reception may be subject to variable network delay. This, however, is regarded by t_y which is, in fact, a random variable. On this account, the constantly repeating course of events can be regarded as a single, static state. Since all events that happen at instant t also happen at instant $t + T$, we disregard time spans greater than T and instead use *phases* $\varphi \in \mathbb{R} \,|\, 0 \leq \varphi < T$, representing the time displacements of periodic events to some (also periodic) reference event. Note that the domain of operations on φ wraps around, e. g. $\varphi + T \sim \varphi$. It can also be viewed as a circle S^1 with radius T/τ with $\tau = 2\pi$ denoting the circle constant. Chapter 6 will deal with parameter spaces that involve several phases and generalizes S^1 to an n-torus $\mathbb{T}^n$. For a visualization refer to Figure 5.4.2 on page 76.

In order to facilitate a terse formulation of expressions in this modulo[5] space we define two operators[6],

$$b \underset{T}{\ominus} a := (b - a) \bmod T, \text{ and} \tag{3.4.1}$$

$$a \underset{T}{\oplus} \varphi := (a + \varphi) \bmod T. \tag{3.4.2}$$

(3.4.1) is used to denote the *phase offset* between two events A and B occurring at times or phases a and b. Accordingly, (3.4.2) denotes the modulo addition of a time or phase a and a phase offset φ. Where it is clear from the context, the subscript T is omitted.

Proposition 3.4.1.

$$b \underset{T}{\ominus} a = 0 \underset{T}{\ominus} (a \underset{T}{\ominus} b)$$

[5] Throughout this work mod denotes the Euclidean definition of the modulo operation according to [22], which always yields non-negative results; it is plotted in Figure 3.4.1.

[6] Alternatively, this wrapping could be expressed in terms of equivalence relations on residue classes. However, instead of using a mixture of equivalence classes and regular variables, I found it less ambiguous and clearer to explicitly state the modulo operation where appropriate.

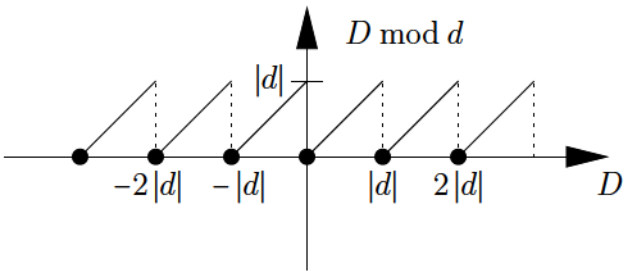


Figure 3.4.1: The Euclidean definition of mod [22]

Intervals

For orientation in such a modulo space we need a useful notation. Given two events A and B that are periodically recurring at a and b, the statement "A comes before B" is of little use, as this implies also the commutation "B comes before A". However, A and B define two distinguished intervals in S^1 namely one from A to B and one from B to A, each in the direction of increasing time:

$$[a,b[_T := \begin{cases} [a,b[& \text{for } a \le b \\ [0,b[\cup [a,T[& \text{for } a > b \end{cases} \qquad \text{with } a,b \in [0,T[$$

Note that $[a,a[$ and hence also $[a,a[_T$ represents the empty set. The complement $\overline{[a,b[_T}$ is the remainder of the domain and, for $a \neq b$, equals the original interval with endpoints swapped:

$$\overline{[a,b[_T} := [0,T[\setminus [a,b[_T = [b,a[_T$$

Such intervals can be viewed as arcs on the domain circle, as shown in Figure 3.4.2.

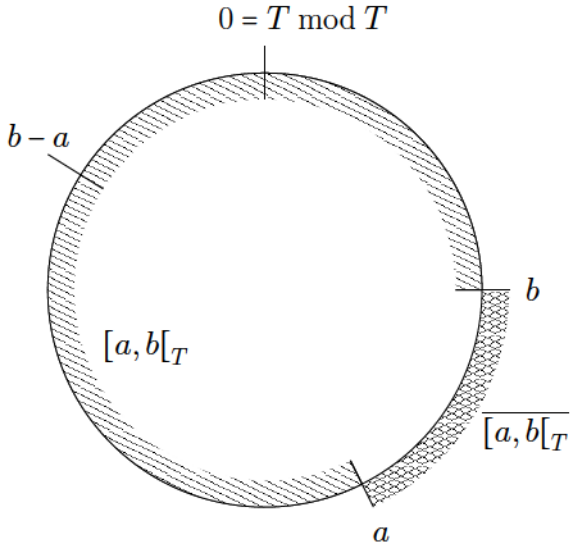


Figure 3.4.2: Intervals in modulo space

At some points it will be useful to formulate inclusion in an interval in S^1 using a system of inequalities. For $x \in [a,b]_T$, two cases need to be considered whilst taking into account the definition of $[a,b[_T$:

$$\begin{aligned} \text{Case } a \le b: \quad & x \ge a \\ & x < b \end{aligned}$$

$$\text{Case } a > b: \quad x \ge a \quad \vee \quad x < b$$

This disjunction can be modeled with a binary case variable i that levers out one of the inequalities:

$$\begin{aligned} x + iT &\ge a \\ x + iT &< b + T \end{aligned}$$

Unsynchronized Events

We let Ω denote a random variable that is distributed according to a uniform distribution across the whole period T. Its probability density function is

$$\text{pdf}\,(\Omega, x) = \frac{1}{T},\ x \in [0,T[\,.$$

It will be used in Chapter 6. Accordingly, we call an event X that takes place at an arbitrary phase $\varphi \in [0,T[$ an independent or *unsynchronized event* . In a statistical context, its phase is a uniformly distributed random variable in D.

$$\text{pdf}(\varphi, x) = \text{pdf}\,(\Omega, x) = \frac{1}{T}$$

Lemma 3.4.2. *Let δ be the difference between two phases φ and θ at which the unsynchronized events X and Y occur . Then δ is also uniformly distributed*[7]*.*

$$\text{pdf}(\theta - \varphi, x) = \frac{1}{T}$$

3.4.2 Implications

As we have seen in the previous subsection, in streaming communications, the order of communication events is undefined. This has strong implications on the communication scheme.

[7] *Proof.* If φ and θ are two independent random variables with probability distributions f and g, respectively, then the probability distribution of the difference $\varphi - \theta$ is given by the cross-correlation

$$f \star g(x) = \int_{-\infty}^{\infty} f^*(\tau) g(x+\tau)\,\mathrm{d}\tau$$

where f^* denotes the complex conjugate of f.In modulo space, $\text{pdf}(\varphi, x)$ and $\text{pdf}(\theta, x)$ are constant functions over the whole domain

$$\text{pdf}(\varphi, x) = \text{pdf}(\theta, x) = \frac{1}{T},\ x \in [0,T[$$

so their cross correlation δ is

$$\begin{gathered} \text{pdf}(\delta, x) = \text{pdf}(\varphi, x) \star \text{pdf}(\theta, x) = \\ \int_0^T \text{pdf}(\varphi, \tau)^* \,\text{pdf}(\theta, x+\tau)\,\mathrm{d}\tau = \int_0^T \tfrac{1}{T} \cdot \tfrac{1}{T}\,\mathrm{d}\tau = \tfrac{1}{T} \end{gathered}$$

□

Periodic Communication Schemes in Two-Way Mode

In [65] a *two-way p-periodic communication algorithm* of length m for a graph $G = (V, E)$ is defined as a sequence $E_1, E_2, ..., E_m$ of some sets (matchings) $E_i \subseteq E$, with $E_i = E_{i+p}$ for all $1 \leq i \leq m-p$ where, for all $e \in E$, $e \in E_i \cap E_j$ implies $i = j$ and where, for each $i \in \{1, \ldots, m\}$ and all $(x_1, y_1), (x_2, y_2) \in E_i$, $\{x_1, y_1\} \neq \{x_2, y_2\}$ implies $\{x_1, y_1\} \cap \{x_2, y_2\} = \varnothing$.

This defines an algorithm suitable for execution on a systolic system [87], in which data is pumped rhythmically and synchronously through a hard-wired processor network, more or less like blood through the body.

In a common peer-to-peer system, communication is inherently asynchronous. This is true not only for different host pairs, but also for individual communication links: Even though the links are full-duplex, sending and receiving need not happen at the same time. A more adequate communication mode could be called *full-duplex node mode*, whereat, during each round, each node may send to one node and receive from one node, possibly but not necessarily the same one. To account for asymmetric links it would furthermore be possible to generalize this to a mode whereat, during each round, each node may send to q nodes and receive from p nodes, akin to the (pR, qS) mode regarded in [50], in which, however, sending and receiving is mutually exclusive. Even so, this modeling approach would be impractical: Using it in a real-life scenario would require not only the knowledge of p and q, that may even be different for every node, but also the network delays for all host pairs because node u sending data to node v will result in node v receiving data from u only after the generally unknown network delay. Also, it is clear that a symmetric gossiping solution is determined by the minimum of p and q alone.

I will therefore disregard constraints at this detail level and only require ordinary full-duplex constraints to hold for the entirety of the period of the communication scheme. Within the period, I will treat transmissions as *transmission events,* singular in time, and disregard, until Chapter 6, the time of reception. Since we seek a solution to the perpetual streaming problem, I will also disregard above margin treatment.

Definition 3.4.3. We let $\mathrm{com}(C, I)$ denote the multiset of one-way transmission events $[(u \to v) \,|\, (u, v) \subseteq E]$ that are initiated during time interval I.

Definition 3.4.4. We define a periodic communication scheme of period T in two-way mode as the unordered set $C = \{E_0, E_1, ..., E_{r-1}\}$ of some sets (matchings) $E_i \subseteq E$, where, for each $i \in [0..r-1]$ and all $(u_1, v_1), (u_2, v_2) \in E_i$, $\{u_1, v_1\} \neq \{u_2, v_2\}$ implies $\{u_1, v_1\} \cap \{u_2, v_2\} = \varnothing$, and for all $e \in E$, $e \in E_i \cap E_j$ implies $i = j$, and, for every interval I of length T,

$$\biguplus_{i=0}^{r-1} [(u \to v), (v \to u) | (u, v) \in E_i] = \mathrm{com}\,(C, I)\,.$$

For every $i \in [0..r-1]$, E_i is called a *logical round* in the two-way communication mode.

It is important to note that the chronology of transmission events $(u \to v)$ is completely undefined, i. e. the rounds may be executed in any order, even interleaved. Since every node can communicate with a different neighbor in each logical round, the scheme is equivalent to a 1-periodic Fr scheme, i. e. a periodic scheme of a single round in which every node may communicate in a full-duplex way with r neighbors. The grouping into logical rounds, which originally served to express adherence to F1 communication constraints, is not strictly necessary. However, since the actual communication scheme will attach further meaning to each logical round, we will retain this notation.

Communicated Payload

The communication scheme according to Definition 3.4.4 only defines who communicates with whom; so far, the content of the communication was left open. To this regard, we define the four symbols Φ, Θ, Ψ and Υ, which we will need later on to define the actual aggregation scheme.

Definition 3.4.5. We let $\Phi(u,v)$ denote the set of information sources that are included in messages sent from u to v . For vertex pairs (u,v) that are not part of the communication scheme, it is defined as the empty set:

$$(u,v) \notin C \;\Rightarrow\; \Phi(u,v) = \varnothing$$

Definition 3.4.6. We let $\Theta(u,v)$ denote the set of information sources received by node u from node v. Plainly, it equals the set of information sources sent by node v to node u.

$$\Theta(u,v) = \Phi(v,u) \tag{3.4.3}$$

Definition 3.4.7. Analogous to Φ and Θ, which are sets of symbols, we define $\Psi(u,v)$ and $\Upsilon(u,v)$ to denote, the aggregated information sent/received by node u to/from node v, respectively.

$$\Upsilon(u,v) = \Psi(v,u) \tag{3.4.4}$$

Periodic Gossiping Communication Scheme in Two-Way Mode for the Fully Allocated Hypercube

Definition 3.4.8. In accordance with Definition 3.4.4 we define the periodic gossiping communication scheme for the fully allocated hypercube as the unordered set $C = \{E_0, E_1, \ldots, E_{L-1}\}$ of sets of vertex pairs that exchange information with

$$E_\ell = \{(u,v) \,|\, v = \mathrm{nb}(u,\ell)\}$$

being regarded as the communications corresponding to logical communication round ℓ or, equivalently, network level ℓ, denoted $Ł_\ell$.

Hypercube Dissemination Scheme

The exchange of information sources is expressed in terms of Φ:

Definition 3.4.9. For gossiping on the fully allocated hypercube we define the set of information sources that are included in messages sent from u to $v = \mathrm{nb}_2(u,\ell)$ as

$$\Phi(u, \mathrm{nb}(u,\ell)) = \begin{cases} \{u\} \cup \{\Theta(u,s) \,|\, s = \mathrm{nb}(u,r) \\ \qquad\qquad \wedge\; r \in [0..\ell-1]\} & (u, \mathrm{nb}(u,\ell)) \in E_\ell \\ \varnothing & \textit{otherwise} \end{cases}$$

Hypercube Aggregation Scheme

Analogous to Φ, the actual information from which the aggregate is formed is defined as follows:

Definition 3.4.10. For gossiping on the fully allocated hypercube we define the information sent by node u to node $v = \mathrm{nb}_2(u, \ell)$

$$\Psi(u, \mathrm{nb}(u,\ell)) = \begin{cases} f\Big(\{g()\} \cup \\ \quad \{\Upsilon(u,s) | s = \mathrm{nb}(u,r) \wedge r \in [0..\ell-1]\}\Big) & \big(u, \mathrm{nb}(u,\ell)\big) \in E_\ell \\ \bot & otherwise \end{cases}$$

The final aggregation result is evaluated at all nodes u from all received information along with the node's own sensor data:

$$\mathrm{agg} = e\Big(f\Big(\{g()\} \cup \{\Upsilon(u,r) | r \in [0..\ell-1]\}\Big)\Big) \tag{3.4.5}$$

Some applications require an aggregate that excludes the node's own sensor data. This applies, for instance, to audio conferencing (2.6.1), where each participant wants to hear everybody else but not his own voice:

$$\mathrm{agg}' = e\Big(f\Big(\{\Upsilon(u,r) | r \in [0..\ell-1]\}\Big)\Big) \tag{3.4.6}$$

The hypercube aggregation scheme is illustrated in Figure 3.4.3. The schematic figure shows how each node's preprocessed information, represented by a filled rectangle in the leftmost column, is aggregated and disseminated to all other nodes. The eight rows, denoted n_0 to n_7, represent the eight nodes of the network. Performing of the merging function f is symbolized by the "+" operator.

The overall aggregation scheme constitutes a directed acyclic graph (DAG), with each node forming a disjoint subgraph thereof. Figure 3.4.4 shows the dissemination DAG for a regular network of 4 nodes.

Within each period, each node needs to perform the tasks shown in Figure 3.4.5. The use case representation expresses freedom in the execution order of the node's tasks. Whereas for one-shot gossiping at least the order of send operations is defined through causality (i. e. each node can only send information it has previously received), for continuous gossiping in modulo-time it is not: All information is potentially available from the previous period. The resulting freedom of interpretation and its implications on overall latency are the subject of Chapter 6.

Periodic Gossiping Communication Scheme in Two-Way Mode for the Incomplete Hypercube

Above we have only considered the case of periodic gossiping on the fully allocated hypercube, where the communication links correspond directly to the edges of the hypercube and the communication scheme is straightforward. But what happens when nodes are removed from the hypercube?

Theorem 3.4.11. *An instantaneous node failure in a previously complete L-level network with 2^L nodes will, until corrective action is taken, result for 2^i nodes in the loss of 2^{L-i-1} sources with $i \in [0..L-1]$.*

Proof. Let u denote the failing node. Let $j = L - i - 1$. Then $i = L - j - 1$. According to Definition 3.4.9, every neighbor $v = \mathrm{nb}(u, j)$ has obtained information from 2^j sources via u and disseminated to further $2^{L-j-1} - 1$ nodes via $Ł_{j+1} \cdots Ł_{L-1}$, i. e. 2^{L-j-1} including itself. Substituting j for i yields the proposition. □

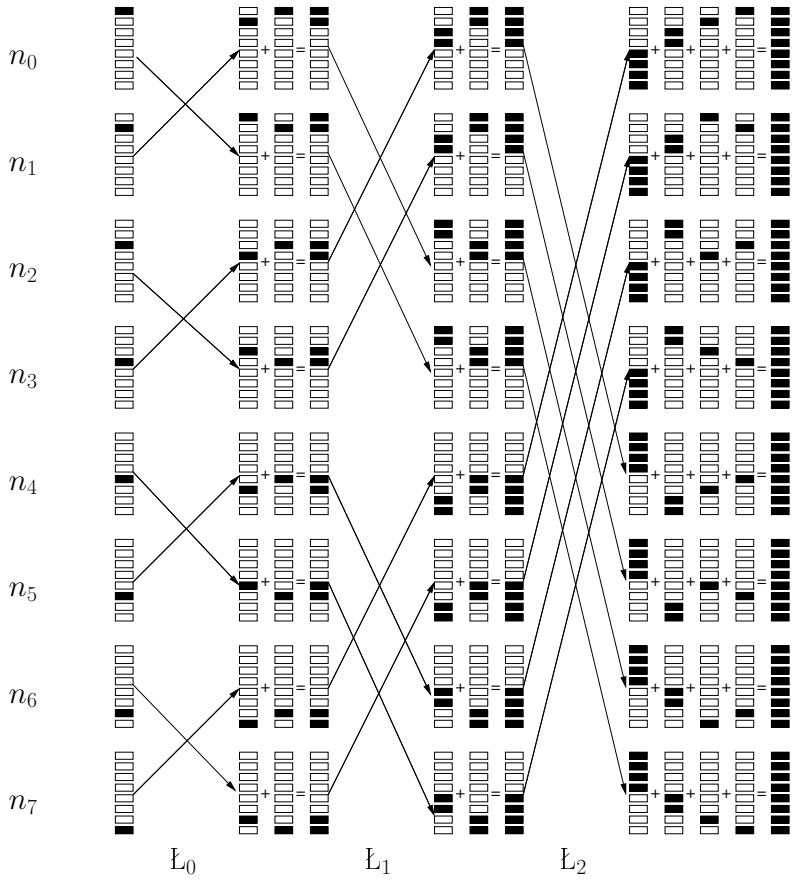


Figure 3.4.3: Hypercube aggregation scheme for $m = 3$. Each open (filled) rectangle represents one piece of information that is missing (present).

How can the gossip scheme be repaired without re-adding a node? When removing one or more nodes from a fully allocated hypercube, two cases can be distinguished:

- If one or more nodes are removed from the top of the ID space, the result is an incomplete but compact hypercube.
- If the nodes being removed are not at the top of the ID space, the resulting ID space will have holes or gaps; the hypercube will be called *porous*.

Although the proposed scheme works equally for both cases, we begin by regarding the more intuitive first case. The incomplete but compact hypercube was characterized in Subsection 3.3.2 as a series of complete hypercubes of descending sizes. It stands to reason that within each of the hypercubes, the gossip scheme according to (3.4.8) can be applied to facilitate gossiping within the respective hypercubes. Consider nodes 8 and 9 of Figure 3.3.2, which form a complete hypercube of size 2. In addition to their mutual Ł$_0$-connection, they are connected to Ł$_3$-neighbors 0 and 1, respectively, to whom they both send their H_1 aggregation result. Nodes 0 and 1 will include this data from 8 and 9 in transmissions to their Ł$_1$ and Ł$_2$-neighbors, but not Ł$_0$-neighbors. (If node 0 would send data from node

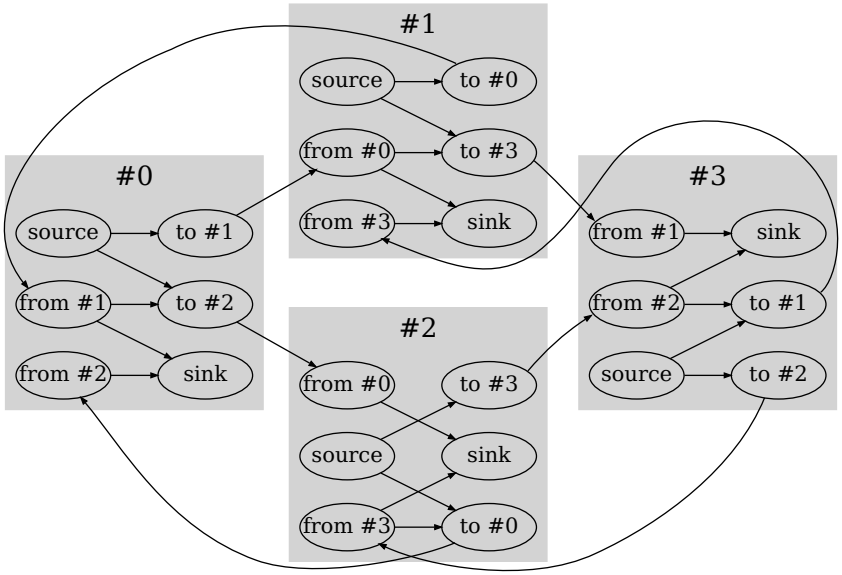


Figure 3.4.4: Dissemination DAG for a regular network of four nodes

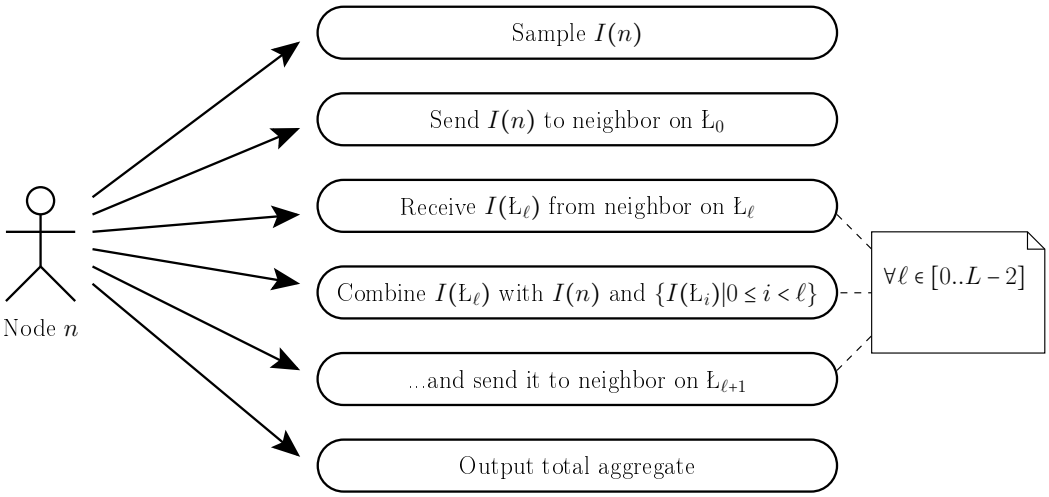


Figure 3.4.5: Activities performed by each node in each period

8 to node 1, it would be duplicated because node 1 receives the data from node 9, too.) Likewise, the data that nodes 0 and 1 receive from their Ł$_1$ and Ł$_2$-neighbors (but, again, not Ł$_0$-neighbors), they send to nodes 8 and 9, respectively. The corresponding dissemination graph is shown in Figure 3.4.6. Note that although nodes 8 and 10 are neighbors with regard to the network topology, shown in Figure 3.3.3, they do not exchange information.

To generalize this scheme to the more general second case of porous hypercubes, two per-node attributes are introduced:

- The *summit* of node u (written $\hat{u}$) denotes a node's largest complete containing

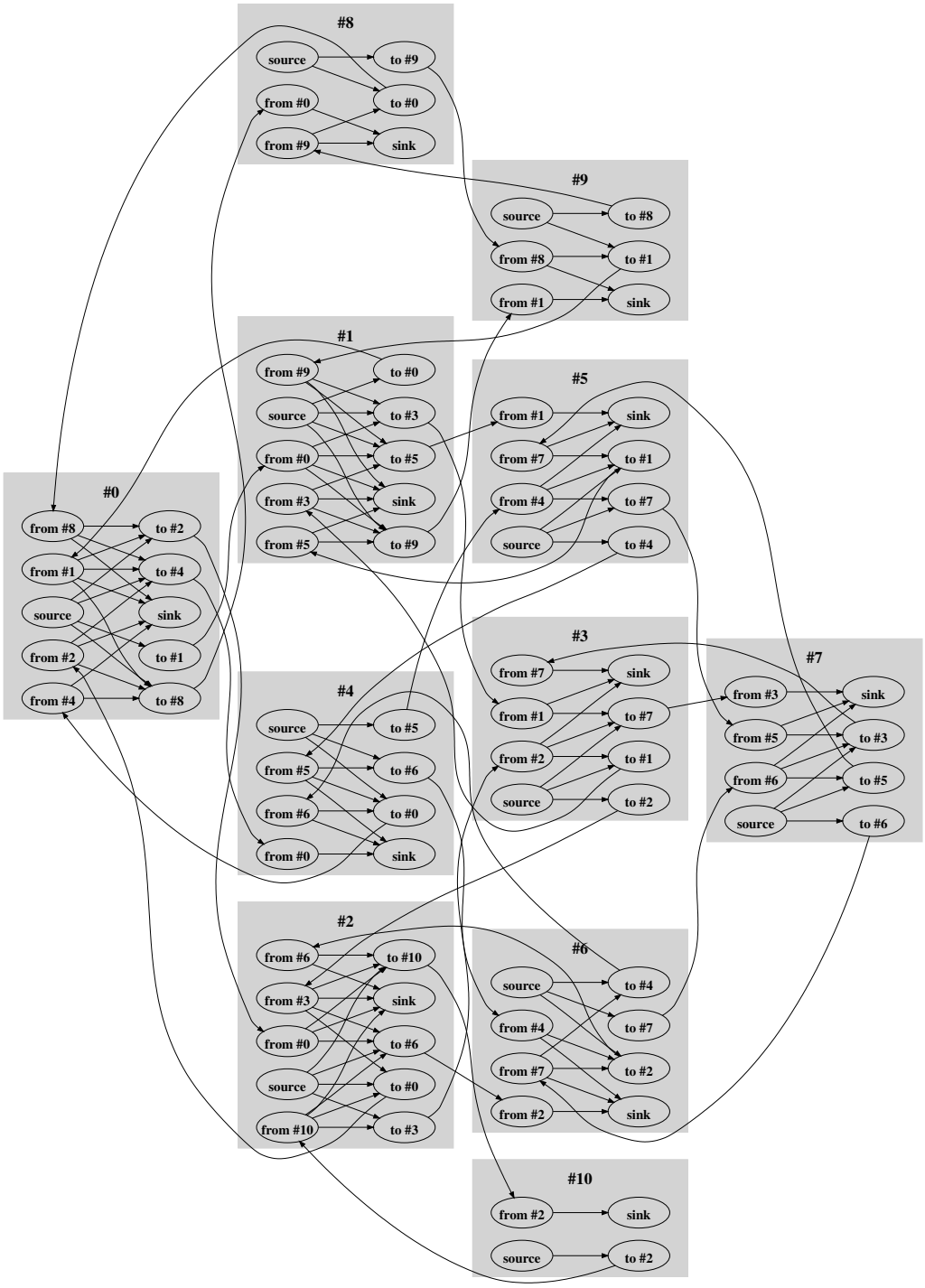


Figure 3.4.6: Dissemination DAG for incomplete allocation of 11 nodes

hypercube. The term becomes obvious from Figure 3.3.2.

- The *maxlevel* of node u (written $\overline{u}$) denotes the highest level ℓ for that $\mathrm{nb}(u,\ell)$ exists.

With these, it is possible to define the scheme as follows:

Definition 3.4.12. We define the periodic gossiping communication scheme for the incomplete hypercube as the unordered set $C = \{E_0, E_1, \ldots, E_{L-1}\}$ of sets of vertex pairs that exchange information with

$$E_\ell = \left\{(u,v) \,|v = \mathrm{nb}(u,\ell) \wedge \left(\ell \leq \min(\hat{u},\hat{v}) \vee \ell = \overline{u} \vee \ell = \overline{v}\right)\right\}$$

being regarded as communications that correspond to logical communication round ℓ or, equivalently, network level ℓ, denoted $Ł_\ell$.

Definition 3.4.13. Accordingly, for gossiping on the incomplete hypercube we define the set of information sources that are included in messages sent from u to $v = \mathrm{nb}_2(u,\ell)$ as

$$\Phi(u, \mathrm{nb}(u,\ell)) = \begin{cases} \{u\} \cup \{s | s = \mathrm{nb}(u,r) \ \wedge \ r \neq \ell \\ \qquad\qquad \wedge \ \ell \geq \min(r+1, \hat{s})\} & \left(u, \mathrm{nb}(u,\ell)\right) \in E_\ell \\ \varnothing & \textit{otherwise} \end{cases}$$

Naturally, $\hat{u}$ and $\overline{u}$ are unknown to node v. Hence, each message from node u to node $v = \mathrm{nb}(u,\ell)$ is annotated with $\overline{u}$ and $\min(\ell+1, \hat{u})$, called the *disposition level*. The recipient node v will use received information only for messages it sends on levels *greater or equal* to the disposition level. In no case it will return the information to the original sender. This ensures proper dissemination in incomplete hypercubes whilst preventing redundant information paths. An imperative formulation of this scheme can be found in Algorithm 7.1 on page 122.

3.5 Latency Overview

On its way from its source to its destination, each data packet experiences various delays, summing up to the total latency[8]. This section provides a brief outlook on the more detailed considerations of latency that follow in the subsequent chapters.

In overlay networks that use packet switched communication, delay falls into different classes, which are described below.

3.5.1 Network Delay

The time span between the commencement of sending of a packet by one node and the instant it becomes available at its neighbor node is called *single hop network delay* or simply *hop delay*. It consists of transmission delay, propagation delay, processing delay, and queueing delay [124, 18, 72]. Due to network locality, hop delay differs across host pairs and usually is not even symmetric.

In the following chapter, the source of network delay is explained in detail, along with its main characteristics in the context of peer-to-peer networks, and how these can be modeled and simulated.

[8] Throughout literature delay and latency are used with slightly different connotations though generally synonymously. In the remainder of this work, I will use the term delay, when referring to a contiguous time span that I treat as atomic, whereas by latency I mean an aggregated sum of delays.

Important from an application's perspective is the time by which a packet will have arrived with a high probability. We let $\chi_{n_1,n_2,q}$ denote the q-quantile of the delay it takes to send a (fixed-sized) packet of samples from node n_1 to node n_2, that is, including all network and processing delay. The choice of p determines the size of the network jitter buffers used to compensate belated data packets. The size of those buffers is a compromise because it defines the maximum lateness the buffer can compensate and the additional delay it causes.

The sum of all hop delays along an overlay path is referred to as *multi-hop network latency*. Note that overlay paths have different lengths as described in Subsection 3.3.2. Hence, across different pairs of overlay hosts, multi-hop network latency exhibits an even greater variance. It is regarded in Subsection 6.1.1.

3.5.2 Wait Delay

The sojourn or bide time [96] of a packet, i. e. the time spent waiting, e. g. after arriving at one node and before being relayed to another, is called *wait delay*. The sum of all wait delays along an overlay path is referred to as *wait latency*. Overall wait latency strongly depends on the nodes' timing mode accounting for the times at which each node sends each packet. It is examined in detail throughout Chapter 6.

3.5.3 Input Delay

Sampling one packet length of consecutive samples on the source node obviously adds a delay of one packet length: By the time the last sample of one packet is sampled, the first sample is one packet length old.

3.5.4 Output Delay

In some applications the destination node will serialize and output (*stream*) the received data one sample after another, e. g. playback audio data. This does *not* per se add additional delay: Admittedly, by the time the last sample of a packet is played, it has indeed aged for one packet length in the output buffer. However, this delay is already accounted for with the input delay because the last sample did not age in the input buffer. Still, the output system (e. g. sound system) may add extra delay for mixing, re-sampling, digital-to-analog conversion, etc.

3.5.5 Local Processing Delay

The local processing delay comprises time needed for any transport compression/decompression, encryption/decryption and the performing of the data aggregation functions e, f, g (see Section 2.3). Commonly, it is proportional to the data packet's size and inversely proportional to the nodes' processing power.

3.6 Related Work

In the spirit of the hypercube or Hamming metric, I will below consider a piece of work related iff it differs from the proposed approach in exactly one dimension, with the dimensions being *topology*, *communication task*, *communication mode*; and the proposed approach being the vector (*hypercube*, *gossiping*, *full-duplex/peer-to-peer*).

3.6.1 Other Gossip Topologies

For a comprehensive theoretical discussion of gossiping on other network topologies, the reader is referred to [65], summarized in Subsection 2.5.3. Also for practical applications various topologies have been proposed:

Complete Graph

The complete graph topology is popular for its simplicity and robustness. At this, the usual gossiping scheme employed with the complete graph is not the optimal F1 scheme (for which a hypercube topology would really be sufficient) but rather a F* scheme where every node sends to every node. While this incurs an effort of $\mathcal{O}(n)$, for some applications, particularly with very small network sizes, it may be a viable alternative.

In the context of audio conferencing, it was first proposed as *full-mesh conferencing* by [93]. In [76] the approach was lifted to an auditory virtual environment and "new voice communication medium, which the author calls VOISCAPE". Because the complete graph is only feasible for a few number of nodes, in MUTUALCAST [94] only a small number of super nodes form a fully connected mesh that the other nodes connect to. In RAVITAS, "a Realistic Voice Chat Framework for Cooperative Virtual Spaces" [152], a complete graph is established only for participants whose avatars are within the same hexagonal cell.

Bus / Multicast

Where multicast is available, each node needs only to send to the multicast channel. This is for instance used by MIMAZE [19], a multiplayer game developed at INRIA. Because of the general unavailability of multicast, it has so far played a subordinated role.

Tree

When it comes to more substantial network sizes, the manageable tree topology achieves acceptable gossip performance despite being suboptimal.

In [126, 127] a periodic gossiping approach called *Distributed Partial Mixing* is presented. The approach comprises combining only some of the streams and transporting other streams separately where bandwidth permits. Their single tree approach, however, seems to be aimed at scenarios with peers behind mixer network components at network boundaries, e. g. corporate gateways or dial-up servers rather than mesh networks like the Internet and individual peers.

PEERTALK, in an approach called *decoupled distributed processing* [58], splits the audio stream processing into two phases, an aggregation phase that mixes audio stream of all active speakers into a single stream via a mixing tree and a distribution phase that distributes the mixed audio stream to all listeners via a distribution tree.

The tree topology is of especial importance to decentralized data fusion because it allows to overcome *data incest*, i. e. the fusion of estimates that were descended from the same data and are therefore not conditionally independent any more. In [57], the problem of double counting is first considered for different topologies (fully connected, tree connected and networks with multiple paths), and the concept of *Channel Filters (CF)* for the removal of common information in tree connected topologies is developed.

2-Tree

[143] proposes a 2-tree topology, that is more resilient to node failures, which in 1-tree or singly connected networks cause a separation of the network. While latency-wise not as efficient as the hypercube topology, the mesh topology of the 2-tree network appears to be a particular fit for wireless (ad-hoc) sensor networks, where each node can contact only those of the others that are in its vicinity.

Path

[137] proposes two methods based on independent cliques whereas "each clique receives a partial estimate from its preceding clique, and updates the partial estimate with local observations made within the clique. Then, it forwards the updated estimate to the next clique". This pipelined distribution is scalable with regard to bandwidth but leads to a high latency. The authors' second algorithm entails forwarding the estimates to a fusion center to obtain the final output.

3.6.2 Other Communication Tasks

The related solutions to the broadcast problem and the accumulation problem in two way mode on the hypercube can be derived as trivial special cases of the solution to the gossip problem. For both of these lesser problems a tree topology would be sufficient. Using a hypercube topology anyway may, however, present some advantages, as pointed out below.

Broadcasting

HyperCuP [134] is a hypercube shaped decentralized peer-to-peer network that offers optimal times for multipoint search or broadcast queries. Any node can be the origin of a broadcast in the network. Node failures are countered by neighboring nodes covering for failed nodes.

A noteworthy approach is taken by the Julia content distribution network [14, 13] that solves the broadcasting problem by first splitting the file to be distributed into pieces and sending one piece to each consumer, followed by a hypercube gossiping phase where all consumers exchange the pieces they have received.

In [43] degree and time bounded broadcast networks are examined. For broadcast time equaling degree, the hypercube topology is optimal.

Accumulation

Accumulation is a dissemination problem frequently encountered in sensor networks when the information of all sensors needs to be accumulated at one node, usually some sort of base station. An important scenario being regarded is that of wireless ad-hoc sensor networks. At this, communication between arbitrary nodes is, if not impossible, oftentimes uneconomic. According to Frii's transmission equation, transmission power requirements increase quadratically with distance between sender and receiver. For this reason, most approaches employ localized path, tree or mesh topologies instead of assuming free communication between arbitrary nodes. Although for accumulation a tree topology is sufficient, a hypercube can provide fault tolerance and load balancing, which is of especial importance for battery powered sensor nodes. The hypercube based approach presented in [33] employs variable subsets of the full gossiping scheme exercised in this dissertation in order to reduce energy consumption and latency.

3.6.3 Other Communication Modes

Before the emergence of communication-centric, asynchronous distributed computing and peer-to-peer systems, one of the main topics in computer science was the study of parallel algorithms and architectures [65]. In [87] algorithms and architectures suitable for Very Large Scale Integration (VLSI) are characterized. The regarded communication mode is that of *systolic communication*. At this, data is pumped rhythmically through a processor network similar to the pulsed flow of blood through the body. In *one-way pipeline mode*, data may flow in only one direction, whereas in *two-way pipeline mode* it flows in both directions simultaneously. The networks considered are linear, square, and hexagonal arrays, as well as trees and shuffle-exchange networks. [65] restricts systolic schemes by allowing an edge to be active only once per cycle of length p, at which the scheme is called *p-periodic*.

The actual design, building and utilization of hypercube based parallel computers began in the early 1980s. Notable specimen, described in Chapter 2, were the Caltech Cosmic Cube [136], the Intel IPSc [75], and the nCube [61].

In such a distributed memory multiprocessor, a communication transaction is an interplay of both hardware and software activity. The H1 model, described in Subsection 2.1.4, makes the weakest assumptions about both hardware and software capabilities, whereas the full-duplex simultaneous or F* mode corresponds to the use of two-way pipeline processors as described in [87]. In [50] the gossip problem is considered for the (pR, qS) communication mode, whereat during each round each node can receive information from at most p nodes or can send information to at most q nodes. Specifically, for $(1R, 1S)$, i. e. H1 mode, the lower bound of $1.44 \log_2 N$ rounds is proven for hypercubes and complete graphs alike. The same is independently proven by [86]. Furthermore, the H1, F1, and F* bounds are compared with the actual performance of algorithms on the nCube, which is shown to lie somewhere in between the H1 and the F1 mode. In [85] hypercube gossiping is regarded in the half-duplex simultaneous or H* communication mode, where each node can participate in an unlimited number of half-duplex communication activities at each time step, and in the H1 mode, also called half-duplex pairwise model, whereat each node can participate in just one half-duplex communication event at each time step. For H1 the upper bound of $1.88k$ rounds for k-dimensional hypercubes is proven.

[96] introduces *perpetual gossiping* as a new information dissemination problem in which gossiping is to occur continuously but with restricted use of the network. The communications used by the gossip process are limited to k concurrent full-duplex calls per time unit to allow the network to be used for other purposes.

More recently, in [51] a family of half-duplex gossiping algorithms is presented. Its members share the same structure but differ—as a function of a combinatorial parameter—in the strategy used to discipline the right to transmit. Gossip complexity can thus be tuned in between quadratic and linear time.

Chapter 4

Modeling Latency in Peer-to-Peer Networks

Contents

One of the main contributions of this dissertation is the characterization and minimization of latency in periodic hypercube gossiping, presented in Chapter 6. Subsequent chapters attend to the implementation and application of these methods. The performance evaluation of these contributions clearly requires a network model that is, on the one hand, an admissible simplification of the true behavior of the Internet and, on the other hand, efficient enough to facilitate extensive simulations with regard to the number of hosts and to simulation time.

Performance evaluation of peer-to-peer (*P2P*) systems can usually only be achieved by means of simulation. At this, network simulators employ network models to reproduce the

characteristics of the network, which are being regarded as a given. When designing and evaluating large scale distributed systems, the behavior of the network needs to be carefully taken into account, especially for P2P streaming applications, which revolve around communication. A network model that does not honor fundamental characteristics of the network will likely lead to misleading simulation results. However, there is a limit to the accuracy of the model. The simulation of typical P2P systems involves a considerable number of nodes and exchanged messages, especially for streaming applications. Hence, due regard must be given to the efficiency of the model. Finally, an underlay network model should lend itself to convenient configuration and parametrization from data that can be obtained in representative amounts.

All in all, efficient, realistic, and usable network models are fundamental to simulating distributed systems and protocols. A network model used for simulation and evaluation of P2P streaming systems should account for proximity, queueing delay, and jitter, yet compute in little time, scale well with respect to the number of hosts, and require no elaborate configuration of parameters.

In this chapter I present an efficient network model as a hybrid approach in the sweet spot between mere analytics and extensive simulation. It is based on measured data and exhibits realistic network behavior by providing location- and load-dependent latency, jitter and packet loss samples for connections between arbitrary hosts world-wide. This makes it especially well suited for the simulation and evaluation of P2P streaming systems. The model features only linear computation and memory requirements with respect to the number of hosts. It benefits from the finding that network delay loss can be conveniently divided into a core network part and a stateful local Internet access part: The former is location- and distance-dependent but from the user's perspective stateless, whereas the latter is stateful. The core network characteristics are modeled using global network positioning (GNP), i. e. by embedding all hosts into an n-dimensional delay space and interpolating delays from their distance before adding regional jitter. The local characteristics are obtained from two queues modeling the send buffers within the local uplink[1] and downlink. The evaluation shows that the simulated delay behavior under varying load much closer approximates real network delay measurements than other models without topology modeling, such as OverSim's Simple-Underlay [8], while still requiring only moderate resources.

The parametrization of the model is established from co-operative HTTP delay measurements, exhibiting a broad range of network performance observations. The GNP coordinates are computed using a novel parallelized relaxation algorithm based on Vivaldi [38] that facilitates the computation of GNP coordinates without depending on explicit landmarks.

The remainder of this chapter is organized as follows: Section 4.1 points out the need for realistic, efficient and usable network models. Section 4.2 illustrates the structure of the Internet and its effect on the composition of network delay. Section 4.3 outlines various approaches to modeling the Internet and end-to-end delay. In Section 4.4, the DELFOI network model is developed. Section 4.5 shows how the model is parametrized from large scale HTTP measurements. Section 4.6 concludes the chapter with a summary and a cue to possible future extensions.

[1] The term *uplink* denotes the connection from data communications equipment toward the network core. It is also known as an *upstream connection*. The inverse is called *downlink*.

4.1 The Need for Realistic, Efficient, and Usable Network Models

With increasing complexity, simulation becomes more and more crucial for the evaluation of large scale distributed systems and protocols. Even though testbeds such as PLANETLAB are available, they may not represent the Internet well enough: With most of their hosts connected to well-provisioned research networks, they may lack the heterogeneity and characteristics of access networks present in the commercial Internet [44]. But also the large scale and consequent setup effort of P2P systems often makes simulation the only choice. At this, overlay networks are instantiated on discrete event (network) simulators, which in turn employ (underlay) network models to reproduce the characteristics of the network.

A network model suited for the simulation of P2P networks should realistically reproduce typical network characteristics, be efficient enough to enable simulations with a high number of nodes, and should be easy to parametrize, configure and use.

Realism

When designing and evaluating distributed systems, particular regard must be given to the behavior of the network. Obviously, a simulation result can only be as meaningful as the assumptions that were fixed—knowingly or naïvely—with the choice of network model. Particularly, when evaluating P2P networks, the following characteristics should be considered:

The fact that some peers (e. g. ones that are geographically closer to each other) may experience smaller latency between themselves, is referred to as *network locality* and regarded in several P2P networks [129, 131, 125]. Yet, bandwidth and latency not only differ between pairs of peers but, due to the asymmetry of residential broadband connections, also between directions. But even for a given path, latency is not constant but varies over time—a fluctuation known as *jitter*. Depending on the state of development of the public infrastructure delay, jitter, and packet loss differ drastically for different parts of the world, whereas in turn, within each region, bandwidth varies greatly. Due to queueing effects, latency may also depend on past transmissions.

A network model that disregards these characteristics of the network will likely lead to misleading simulation results, especially for streaming applications with real-time requirements.

Efficiency

Although all of above features can be reproduced with network simulators, for the simulation of large scale P2P networks, the associated effort is prohibitive.

Already, the measurement and parametrization of a sufficiently large portion of the Internet means a considerable effort, let alone the computational complexity and memory requirements for the simulation [59, 78]. The problem is multiplied in the case of P2P streaming applications, whereat every node emits packets with high frequency.

Consequently, the network model in question ought to use shortcuts and optimizations in order to emulate network behavior without simulating it in every detail.

Usability

Besides its computational complexity and memory requirements, a suitable model's configuration and parametrization with realistic data may be non-trivial, too. Tedious configuration and parametrization lead to researchers still choosing rather simple models.

Samples for network parameters may be hard to obtain, even for a small number of hosts [78]. Firewalls that come enabled with access routers nowadays prevent obtaining measurements in representative amounts, e. g. by ping probing random hosts.

A network model should lend itself to convenient configuration without the need to hand-tune parameters. The required measurement data should be obtainable in representative amounts sufficient for large scale simulations.

4.2 Characterizing Internet Delay

We define *(single) hop (Internet) delay* as the time span between the commencement of sending a packet at one node and the instant it becomes fully available at the neighbor node.

To understand the characteristics of this hop delay, it is obligatory to first regard the structure of the Internet.

4.2.1 Internet Structure

Illustrated in Figure 4.2.1, the Internet is composed of many autonomous systems (*AS*), such as networks of enterprises, Internet service providers or universities, that are connected to one or more other AS by boundary routers [141, 124].

An IP packet sent from a typical endpoint with consumer Internet access is first modulated by the DSL or cable modem, traverses an uplink (e. g. ADSL or cable) to the network of the first endpoint's ISP, and is then separated and demodulated by the *digital subscriber line access multiplexer (DSLAM)* or *cable modem termination system (CMTS).* The packet then passes through a *broadband remote access server (BRAS),* that enforces quality of service policies, before it crosses several routers of autonomous systems. At the receiver's access provider, the packet passes again through BRAS and CMTS or DSLAM and is delivered via the downlink to the receiving modem and host.

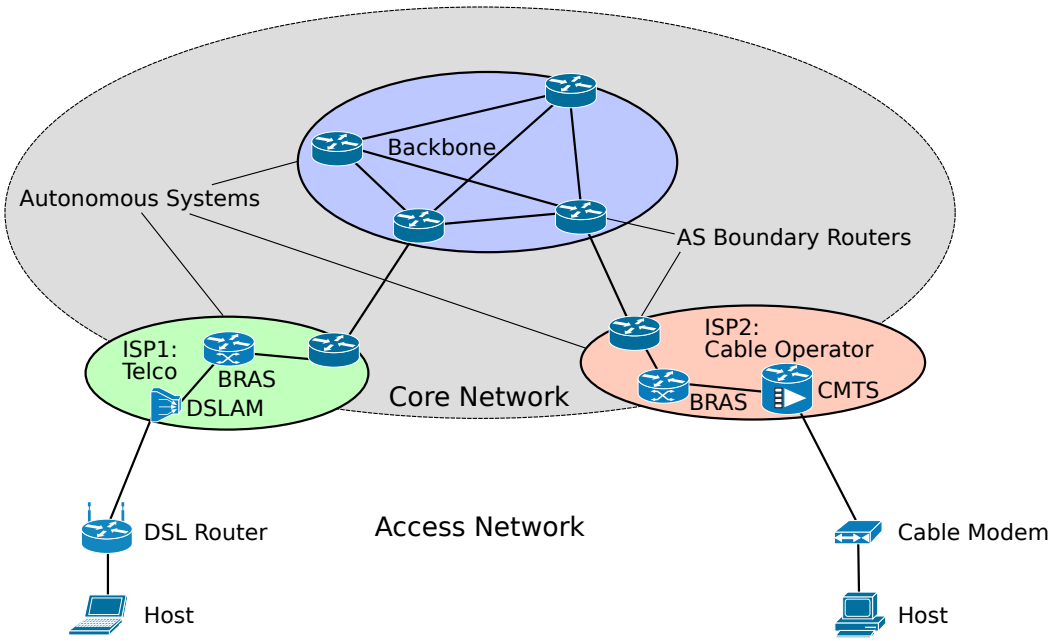


Figure 4.2.1: Structure of the Internet

4.2.2 Delay Components

Given this structure, the causal and spatial composition of delay can be itemized:
Causally, hop delay consists of

- transmission delay (i. e. packet size divided by link speed),
- propagation delay (i. e. physical distance divided by propagation speed of medium),
- processing delay, and
- queueing delay [18, 72].

Spatially, at least three sections with different delay characteristics can be discerned along a consumer-to-consumer network path:

- uplink, i. e. consumer host to ISP (source access network),
- ISP-to-ISP (core network), and
- downlink, i. e. ISP to consumer host (destination access network).

4.2.3 Observations

Spatially, a substantial amount of delay originates in the access network, such as in the case of typical consumer ADSL lines. ADSL ("Asymmetric Digital Subscriber Line"), as the name implies, is characterized by a bidirectional data channel that is asymmetric: The downstream bandwidth is greater than the upstream one. This design is based on the

assumption that the typical consumer in fact basically *consumes* information and the upstream is mainly used to request downstream data. However, with the adoption of P2P systems, this assumption no longer holds. The communication is now among peers; consequently each participant receives and sends data in more or less equal parts. The uplinks of endpoints usually form the bandwidth bottlenecks with noticeable send delay whereas paths within and between AS provide relatively high bandwidths. Transmission delay (i. e. packet size divided by link speed) is defined by this bottleneck, i. e. the minimum bandwidth along the path. Moreover, increased amounts of sent data easily cause congestion before the upstream link of the modem, i. e. queueing delay is built-up by dispatching data packets faster than they can be transmitted through the connection's bottleneck. This can lead to considerable queueing delays in P2P systems. Finally, DSL providers often employ interleaving as part of their forward error correction. At this, the data is rearranged prior to being sent in order to increase the chance of reconstructing the data in the case of bursts of line noise. According to [81], this accounts for another 20–40 ms of delay. As for the core network, the remaining queueing delay obviously also depends on the level of congestion, but is, at least for fully developed regions, considerably smaller and effectively independent from the clients past transmissions. With regard to propagation delay, [78] shows that round trip times are indeed largely proportional to the geographical distance between the hosts. Processing delay on a network path is roughly constant because, independent of packet size, it takes almost the same number of machine instructions to process a packet [72]. Effectively, the resulting end-to-end delay between two hosts, and for packets of a certain size, is characterized by a constant, minimum delay augmented by a variable, supplemental component. The constant portion is determined by transmission, propagation and processing delay. The supplemental component is determined by queueing and media layer retransmissions and commonly referred to as *jitter*.

Figure 4.5.7 shows an estimate of minimal hop delay for packets of 400 bytes between arbitrary hosts in Germany, March 2010. The estimate was obtained using the DELFOI model, developed in the course of this chapter and parametrized with measurement data obtained from `speedtest.net`.

Worst Case Delay

Due to the Internet's heterogeneous nature, end-to-end delay is long tailed [64]. Hence, considering maximum network latency, either over time or over host pairs, is of little use. In theory, IPv4 packets include a time-to-live (TTL) field, denoting the number of seconds after which the packet should be destroyed [68]. As the field is 8 bit large, this allows for a maximum hop delay of 255 seconds. "In practice, very few, if any, IPv4 implementations conform to the requirement that they limit packet lifetime" [42]. In any case, the practical value of this time value is limited. Accordingly, it has been replaced by a "Hop Limit" field in IPv6. Experiments in Subsection 4.4.3 show that several seconds of hop delay can easily be provoked by saturating Internet access links.

4.3 Related Work

Internet delay can be modeled on various layers of abstraction and levels of detail, as realized by the related work summarized below.

4.3.1 Statistical Models

Statistical models condense the behavior of the Internet into a formula. Their advantages are simplicity, minimal computational effort, and virtually no memory costs. In fact, [113] shows that end-to-end round trip times (RTT) can be adequately approximated by a constant plus gamma distribution as long as the time interval over which the distribution is estimated is kept short. Remarkably, this is found to be true for all network paths studied (regional, backbone, and cross-country) and for all observed loads, although the parameters and the accuracy of the approximation vary.

In [63] the use of a single probability distribution together with parametric modeling is generalized into a K-bin histogram feeding a Support Vector Machine (SVM) which is then sampled using the Slice Sampling algorithm.

Statistical models may be well sufficient if the system under evaluation uses a homogeneous portion of the net and relationships between individual network hosts (e. g. proximity) and stochastic dependence of consecutive values are irrelevant. However, it is unsuitable if the system under test depends on locality or (region dependent) jitter.

4.3.2 Global Network Positioning (GNP)

Global Network Positioning [115] uses distance in some Euclidean space to compute delay. The overlay network simulation framework OverSim as described in [8] features a network model called Simple-Underlay that uses the geographical distance between two Earth locations to estimate network delay:

> "For this, each node is placed into a two-dimensional Euclidean space. In addition the node is assigned to a logical access network characterized by bandwidth b_n, access delay d_n and packet loss, so that heterogeneous access networks can be simulated. The end-to-end packet delay d_e of packet P with length l_P between overlay nodes A and B is then calculated as follows
>
> $$d_e = d_1 + \frac{l_P}{b_1} + c \cdot \|A - B\|_2 + d_2 + \frac{l_P}{b_2} \tag{4.3.1}$$
>
> where c is constant."[2]

In [78] the validity of using geographical distance to determine delay is evaluated. It is shown that there is a proportionality of the RTT to the length of the transmission medium, however, for some regions (e. g. Africa, South Asia) the observed delay is much higher than the mere distance suggests. As a consequence, the authors provide a synthetic, higher dimensional distance model that better models the observed delays. To this end, the simulation hosts are placed into the space in such a way that the computed minimum round trip times approximate the measured distance as accurately as possible, based on the approach of [114]. The authors provide both models in their discrete event based P2P simulator PeerfactSim.KOM that allows for simulation of P2P protocols and applications.

The Vivaldi coordinate system [38] augments each (2-dimensional) Euclidean coordinate with an additional positive, directionless "height" value to accommodate access link delays: The Euclidean distance based end-to-end delay estimation is increased by a delay value proportional to the sum of both nodes' height values. While this approach notably increases prediction accuracy over simple 2-D or 3-D coordinates, it does not account for jitter, access link queueing, or delay dependence on packet size.

[2] A more recent version of OverSim now also includes access link queueing as well as flexible GNP dimensions.

4.3.3 Network Simulators

While statistical models try to mimic some statistical aspects of the Internet as a black box from the outside, network simulators model the inner workings of the Internet, including backbone networks and routing protocols. Network simulators operate on a network graph including bandwidth and capacity of links and routers, commonly called *topology*. Notable examples are NS-2/NS-3[3] and OMNeT++ [145] in conjunction with the INET Framework [4].

Obviously, a detailed model exhibits a behavior that resembles much closer the behavior of the real Internet and can be used to reproduce various phenomenons (e. g. packet-reordering, asymmetric routes, packet compression, triangle inequality violations) and, for instance, analyze benefits of access network aware overlay networks [8]. However, the Internet router level topology is very hard to capture because of its huge size and its dynamic changes. Also, the associated simulation effort is considerable [100].

4.3.4 Topology Generators

In many cases the actual topology of the Internet is irrelevant to the evaluation. Instead of simulating the real Internet, topology generators attempt to generate a network graph whose structure resembles the one of the Internet [99]. This obviates the need for huge Internet maps but still leaves the simulation effort. Notable specimen include GT-ITM [26], BRITE [110] and Inet [149].

4.4 DELFOI—A Hybrid Modeling Approach

To answer a question of delay between two arbitrary Internet hosts in the absence of total knowledge is an oracle's task. DELFOI[5] takes an educated guess by combining PeerfactSim.KOM's GNP-based delay and jitter estimation [78] with a simulation model of a local Internet access link.

4.4.1 Assumptions

The DELFOI model rests on the following assumptions in particular:

- The bulk of nodes in large scale P2P systems is comprised of (home) computers connected to the Internet via some kind of access link, e. g. DSL, cable or dial-up modem.
- The access link is the bottleneck in such an end-to-end Internet connection.
- The core network congestion caused by an individual user is negligible.

From these assumptions follows that the delay of an end-to-end connection can be be divided into two characteristic classes:

1. When there is only little load on the access links, the delay is characterized by the sum of the deterministic upstream transmission delay, the stochastic core network delay, and the deterministic downlink transmission delay on the receiving side. This

[3] `http://www.isi.edu/nsnam/ns/`
[4] `http://inet.omnetpp.org/`
[5] Delay Estimation Localized For OMNeT++/INET

agrees with the analysis of [72] and the delay models of OverSim Simple-Underlay [8] and PeerfactSim.KOM.

2. When the access link is under full load, however, significant additional delay can result from packets queueing before the uplink. In this case, end-to-end delay is easily dominated by this queuing delay.

The PeerfactSim.KOM simulator uses the following estimation to model Internet delay:

$$delay\left(H_1, H_2\right) = \frac{RTT_{\min}}{2} + jitter \tag{4.4.1}$$

The minimum round trip time $RTT_{\min}$ is determined by the Euclidean distance in the D-dimensional delay space:

$$RTT_{\min} = \sqrt{\sum_{i=1}^{D} \left(c_{H_1,i} - c_{H_2,i}\right)^2}$$

Jitter is sampled from a log-normal distribution $f\left(x;\mu,\sigma\right)$ with the probability distribution function:

$$f\left(x;\mu,\sigma\right) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma x}} \exp\left(-\frac{1}{2}\left(\frac{\ln x - \mu}{\sigma}\right)^2\right) & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4.2}$$

The parameters μ and σ are adjusted in such a way that both the expectation and the inter-quartile-range value match the target (e. g. measured values).

DELFOI extends this PeerfactSim.KOM estimation (4.4.1) by models of each end's Internet access link. The one-way end-to-end delay for a packet of size l_P thus becomes:

$$\begin{aligned} delay\left(H_1, H_2, l_P, S\right) \quad = \quad & access_{\text{uplink}_{H_1}}\left(l_P, S_{\text{uplink}_{H_1}}\right) + \frac{RTT_{\min}}{2} \\ & + jitter + access_{\text{downlink}_{H_2}}\left(l_P, S_{\text{downlink}_{H_2}}\right) \end{aligned} \tag{4.4.3}$$

Each of the two access links is modeled as two, currently infinite, queues—one for the uplink, one for the downlink—with a specific service bandwidth ($bandwidth_x$) that is proportional to the size of the current packet as well as a host specific constant access network delay ($andz_x$). The delay of access link x takes the queue fill level $queuelevel_x\left(S_x\right)$ into account which depends on the current state S_x of the queue.

$$access_x\left(l_P, S_x\right) = \frac{queuelevel_x(S_x) + l_P}{bandwidth_x} + andz_x$$

4.4.2 Implementation

Figure 4.4.1 shows the architectural overview of the DELFOI model. The overall implementation is based on the discrete event simulation environment OMNeT++ [145]. Within OMNeT++ one particularly notable model/extension is the communication networks simulation package called INET Framework[4], that contains models for various Internet protocols. DELFOI extends the INET Framework by an abstraction—a module called `InternetCloud`—to which any number of `StandardHosts` can be connected by their `PPP` interfaces. `InternetCloud` routes all packets between connected hosts, after delaying them by an amount depending on source and destination address, subject to the

delay model described in Section 4.4. The GNP and jitter delay computation code is based on PeerfactSim.KOM [119] which is a discrete event based P2P simulator written in JAVA. In order to make the PeerfactSim.KOM model usable within OMNeT++, the delay related code portions had to be ported from JAVA to C++ first. The resulting library gnplib[6] can read measurement data files in PeerfactSim.KOM's XML-based file format and provides delay data through the module `GeoLocationOracle`.

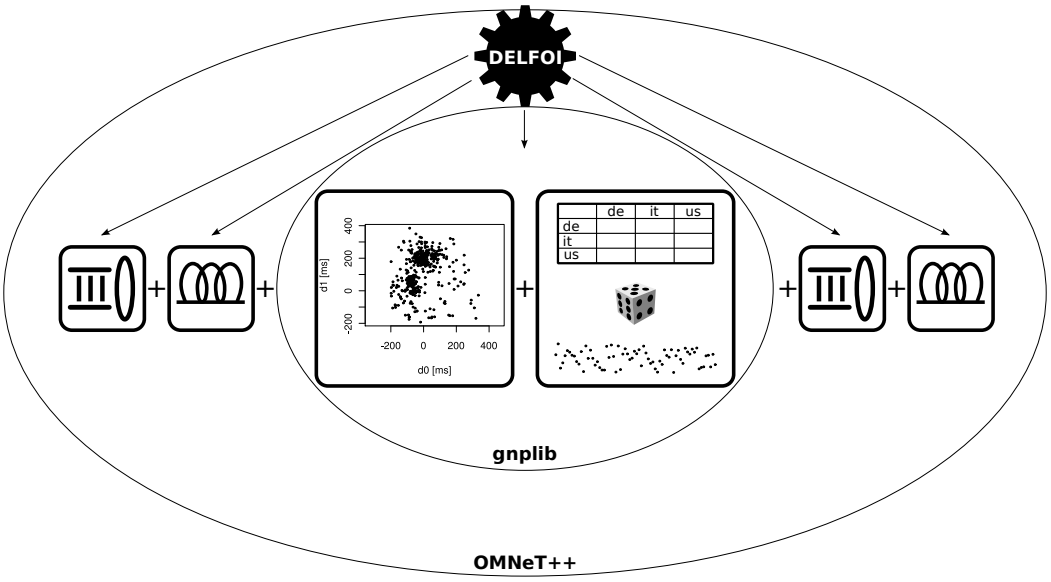


Figure 4.4.1: DELFOI architectural overview

4.4.3 Evaluation

In the following experiments, the accuracy and performance of DELFOI is evaluated. In the experimental setup, one first host, subsequently called *Sender*, that has high speed Internet access, sends ICMP probe packets (*"pings"*) of varying size to a second host, subsequently called *Recipient*, with lower speed Internet access, that sends them back to the first. The packet sizes were chosen arbitrarily but contrasting, without exceeding the MTU. To send the ICMP packets, the PING program from IPUTILS [90] was modified to send these differently sized packets consecutively without additional intermittent delay. At Sender, all packet send and receive events are recorded with the capture tool and packet analyzer TCPDUMP.

Parametrization for this evaluation was done using the results of real-world measurements on reference hosts in Germany having no firewall in the DSL modem. Each host's Internet access speed was measured using the free service provided at `www.speedtest.net`. The measurements produced 93,990 Kbps downstream, 55,958 Kbps upstream for Sender and 2,954 Kbps downstream, 373 Kbps upstream for Recipient. The constant portion of the access link delay for Recipient was determined by measuring the round-trip times of two differently sized ICMP packets sent to the broadband remote access server (BRAS) and extrapolating the time for a zero-sized packet, yielding 44 ms.

[6] `http://sourceforge.net/projects/gnplib`

After concluding the real-world measurements, the model was configured with the above values. The remaining model parameters were obtained from the following real world measurements:

- The host list and the GNP coordinates were derived from RTT measurements performed from August 2007 by the CAIDA Macroscopic Topology Measurements Project[7] as described in [78].
- The region specific jitter measurements were obtained from the PingER Project[8] as described in [78].

Hosts Sender and Recipient were connected via the DELFOI `InternetCloud`. The `GnpNetworkConfigurator` assigns random IP addresses to the hosts from a pool of IP addresses annotated with GNP positions. According to PingER, RTT jitter for packets from and to Germany is distributed with an expected value of 1.68 ms and an inter-quartile range of 2.25 ms. A log-normal distribution was fitted automatically to these values and used as source for random jitter values. The `GnpNetworkConfigurator` randomly chose the pseudo IP address 194.95.250.69 for Sender and 134.96.26.1 for Recipient from the data set. The minimum round-trip time as determined by the `GnpLatencyModel` based on the CAIDA data was 7.43 ms.

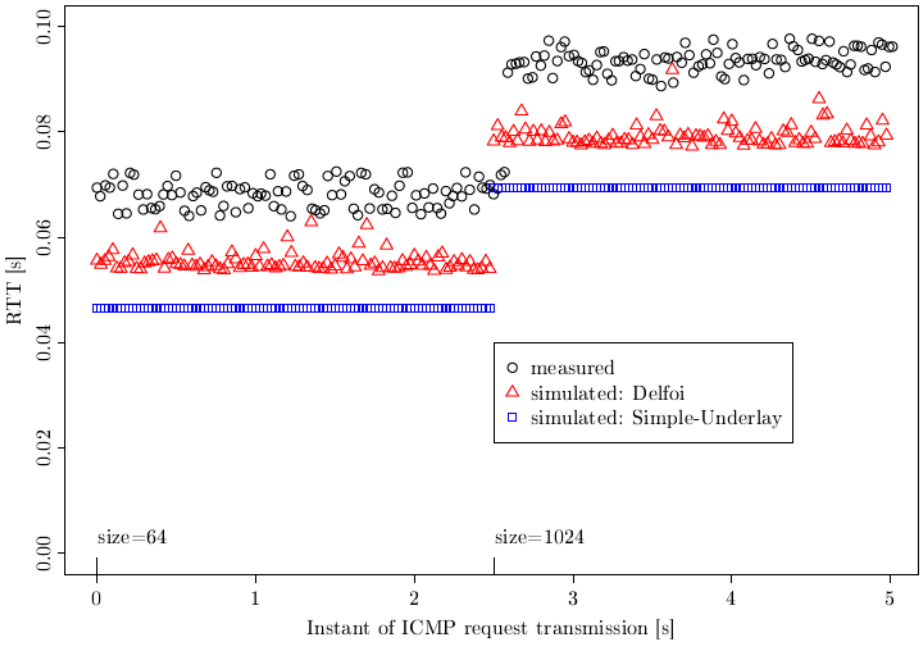


Figure 4.4.2: Round-trip time for 100+100 ICMP packets of sizes 64, 1024 bytes, sent at an interval of 25 ms

In the first run, 100 ICMP packets with a size of 64 bytes were sent, followed by 100 packets with a size of 1024 bytes. The send interval was 25 ms. Figure 4.4.2 shows each probe's round-trip time plotted to its ICMP request packet send instant. The black

[7] `http://www.caida.org/projects/macroscopic/`
[8] `http://www-iepm.slac.stanford.edu/pinger/`

circles show the reference measurements, while red triangles and blue squares represent delays simulated by DELFOI according to (4.4.3) and SIMPLE-UNDERLAY, i. e. (4.3.1) according to [8], respectively. The simulated values of both models adequately approximate the reference values. The constant offset does not surprise, since the hosts were chosen randomly from a pool of hosts in that region. As expected, the increase in packet size at time 2.5 s leads to a corresponding increase in delay. The measured jitter was approximately twice as high as the jitter predicted by PINGER and simulated by DELFOI and more uniformly distributed. The constant delay of the SIMPLE-UNDERLAY model disregards jitter altogether.

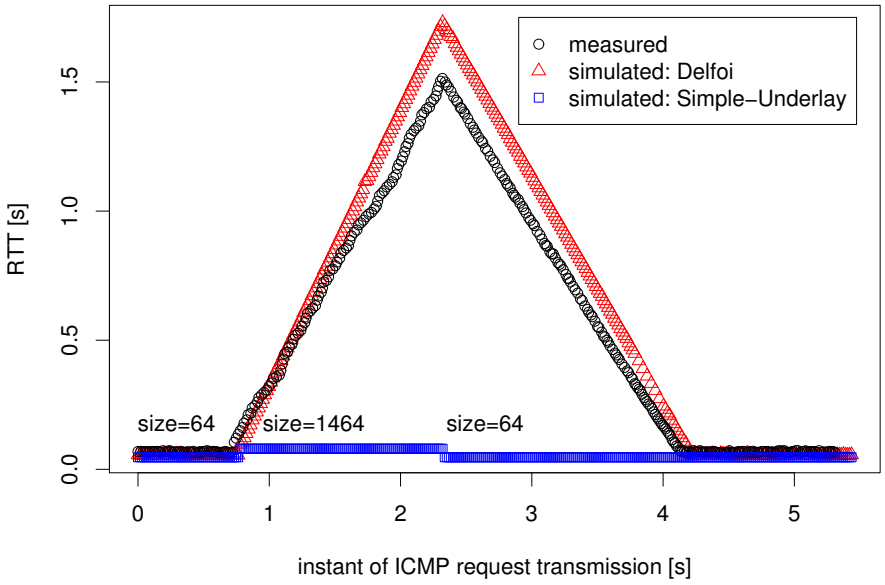


Figure 4.4.3: Round-trip time for 50+100+200 ICMP packets of sizes 64, 1464, 64 bytes, sent at an interval of 15 ms

In the second run, 50 ICMP packets with a size of 64 bytes were sent, followed by 100 packets with a larger size of 1464 bytes, followed by 200 again with a size of 64 bytes. The send interval was lowered to 15 ms. Figure 4.4.3 shows how the measured delay begins to rise rapidly as soon as the packet size is switched to 1464 bytes. The DELFOI model takes this queueing at Recipient's uplink into account and adjusts the delays of the following packets accordingly. Apparently, the uplink bandwidth measured by speedtest.net is somewhat too low, causing a slight overestimation of the buildup. The SIMPLE-UNDERLAY model (4.3.1) takes into account the increased transmission time, noticeable by the step-like increase in delay, but disregards the congestion.

The computational performance of DELFOI was evaluated using the basic "routerperf" example of the INET FRAMEWORK, in which 1000 packets from each of three source hosts are sent to three destination hosts. When the single router was replaced by DELFOI's InternetCloud, the simulation time on an Intel® Core™2 Duo CPU T7500 @ 2.20 GHz

increased by only 30 %.

4.5 Parametrizing the DELFOI Model from HTTP-Measurements

Having established the model structure, we now focus on its parametrization from large-scale measured Internet speed data. For the evaluation in the previous section, the host list and the GNP coordinates were derived from CAIDA RTT measurements and the region specific jitter, from the PingER Project. Internet access link speed parametrizations for the model were limited to a few manual measurements, most of which were severely hindered by firewalls that come with Internet access routers.

The failure to probe these hosts, leaves measurements projects no other choice than to skip these hosts. Because routers are shipped by the Internet providers with firewalls enabled, omitting these hosts results in excluding whole classes of networks with characteristic qualities. Therefore, in order to gain a representative view that includes these customer classes, whose characteristics are of eminent significance to P2P networks in particular, we have no other choice than to rely on the co-operation of Internet users.

4.5.1 Measurement Methodology

The website `www.speedtest.net` offers a free service to measure one's own Internet connection, including upstream and downstream bandwidth and latency. For this, the user's web browser downloads and executes an Adobe Flash program that initiates HTTP connections with one of over 500 speedtest.net servers located all around the globe. The geographically closest server is chosen per default. Speedtest.net uses GeoIP data from MaxMind[9] to determine the user's position and the speedtest.net server closest to her. Because the probing is performed with regular HTTP requests, initiated by the client, probe packets pass through most users' firewalls unhindered.

Round trip time is measured by downloading ten byte documents for ten times, with the lowest value determining the final result. This differs from the standard procedure to "ping" hosts, as TCP/IP is used instead of ICMP. The sizes of the IP packets are roughly 477 and 453 bytes for HTTP GET request and HTTP response, respectively, whereas ping defaults to 86 byte ICMP packets. The use of TCP does not pose a problem as both HTTP messages fit into one segment each. Flow control should not intervene, as the connection is idle before. Finally, the time it takes an Apache web server to serve that 10 byte document may be somewhat higher than the time it takes the operating system to reply to a ICMP echo request but not by much if it is served from memory.

Downstream bandwidth is measured by first downloading some small files from the server to estimate the connection speed. Based on this result, one of several file sizes is selected for the real download test. Presumably this is done to keep the duration of the test roughly constant. The actual download test is then started on several TCP connections at once. We presume, this is to fully saturate the line, as one TCP connection alone usually leaves some "air" due to TCP flow control. According to Ookla, throughput samples are taken at up to 30 times per second and then aggregated into 20 slices (each being 5% of the samples). The fastest 10% and slowest 30% of the slices are then discarded:

[9] `http://www.maxmind.com/`

> "Since we are measuring data transported over HTTP via Flash there is potential protocol overhead, buffering due to the many layers between our application and the raw data transfer and throughput bursting due primarily to CPU usage. This accounts largely for dropping the top 10% and bottom 10% of the samples. We also keep our default test length short for the user experience, and compared to this duration the ramp-up period is fairly significant driving us to eliminate another 20% of the bottom result samples." [1]

The remaining slices are averaged together to determine the final result.

For measuring the upstream bandwidth, again, the connection speed is first estimated and an appropriate data size is chosen. The upstream test is performed via HTTP POST in several chunks and several TCP connections. The fastest half of the measurements is chosen and averaged to eliminate anomalies.

Note that the bandwidth measured this way represents a HTTP payload or "net" bandwidth and does not consider HTTP, TCP, or IP headers. This is perfectly consistent with the latency measurement; however, when using speedtest.net's data for parametrizing IP-level network simulators, appropriate conversion factors must be regarded.

4.5.2 Parametrization Methodology

Recall that the model is split into an access network part and a core network part. Preferably, parametrizations for both parts should be obtained from the same measurement source to avoid mismatches. For the parametrization of the access network part we need measurement connections to servers that are as close as possible to the Internet provider's broadband remote access server (BRAS), so propagation delay, AS routing and queueing delay do not overly distort the access network model. Fortunately, such measurements should be abundant, since speedtest.net selects the geographically closest server by default. For the arranging of the hosts in D-dimensional delay space we need measurement connections to at least $D+1$ different reference servers per host. This requires a certain amount of curiosity from the users, since speed tests with different servers need to be selected manually and may not be in the users' primary focus, at least if they just want to check whether "their Internet is slow". But even if a sufficient amount of users initiated connections to a sufficient amount of reference servers, we also needed the latencies between all reference servers to establish a set of landmarks [114]. In lack thereof, we shall resort to determining the host arrangement in delay space by a modified version of the (centralized) Vivaldi algorithm [38]. Vivaldi models network nodes as points in delay space that are connected by springs, with the spring lengths at rest equaling the measured delay values. The Vivaldi algorithm iteratively iterates over all nodes and moves them some fraction in direction of the total force, thus minimizing the sum of all springs' squared errors equaling the energy of the spring network.

For the parametrization of the upstream and downstream bandwidths of a hosts access link the maximum measured bandwidth values for that host are chosen, as these approximate the physical channel most closely. Deviances from this value originate from either bit errors, resulting in re-transmission, or link utilization by the user, e. g. his operating system or mail program checking for updates or mails, automated attacks by malware from the out- or inside, etc. [105]. For the parametrization I assume all bandwidth fall-offs to be caused by link utilization—a kind of vitiation that can be well reproduced by the DELFOI model.

Figure 4.5.1 illustrates how the measured latencies (represented as distances on the number line) between one client host C and two servers S_1, S_2 are split into access network

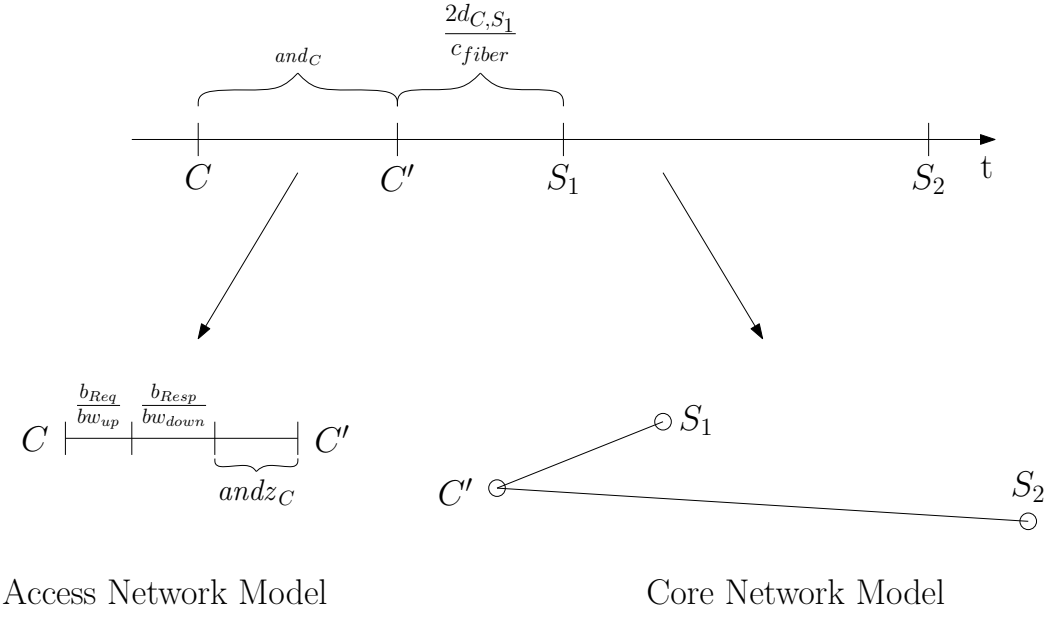


Figure 4.5.1: Separation of delay

and core network portions. We select the client's minimum round trip time to its closest server (S_1 in this case). From this we subtract twice the distance between host and reference server, divided by the speed of light in fiber, as this portion is the lower bound for the delay between BRAS and reference server. The remaining delay portion is considered position independent (but host specific) access network delay and_C:

$$and_C = rtt_{C,S_1} - \frac{2d_{C,S_1}}{c_{\text{fiber}}}$$

It is subtracted from all client/servers delays before applying the Vivaldi algorithm to the remaining (distance dependent) delays $\left\|\overline{C'S_1}\right\|, \left\|\overline{C'S_2}\right\|$.

For the parametrization of the access network model, the access network delay and_C is decremented by delay due to the size of the probe packets, which depends on the bandwidths, yielding the constant time required to send a (hypothetical) zero-sized packet to the BRAS and back:

$$andz_{C_{\text{uplink}}} + andz_{C_{\text{downlink}}} = andz_C = and_C - \frac{bytes_{\text{request}}}{bandwidth_{\text{uplink}}} - \frac{bytes_{\text{response}}}{bandwidth_{\text{downlink}}}$$

4.5.3 Results

Our data set from speedtest.net consists of 2.9 million measurements collected between March 26th–28th, 2010. Of the one million users in the data set, 60,000 users initiated a total of 270,000 connections to at least three different servers, of which the 230,000 unique user–server combinations were selected.

Figure 4.5.2 shows the cumulative distribution of the maximum measured upstream bandwidth. Half of the users, demarked by Q1 and Q3, have a sharply defined upstream bandwidth between 300 and 1100 kbps.

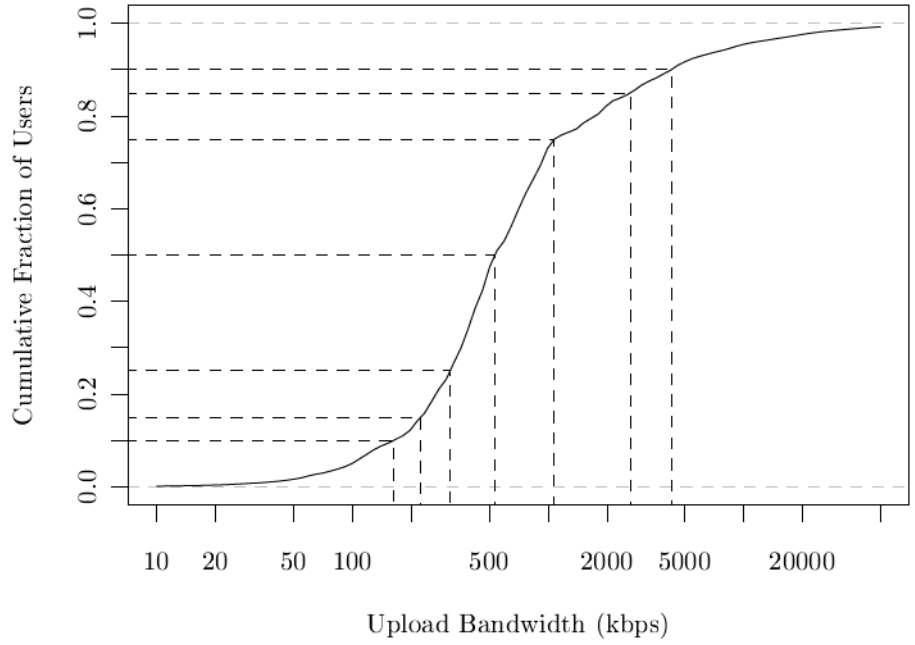


Figure 4.5.2: Cumulative distribution of upstream bandwidth. Markers at 10 and 15 %, Q1, median , Q3, 85 and 90 %

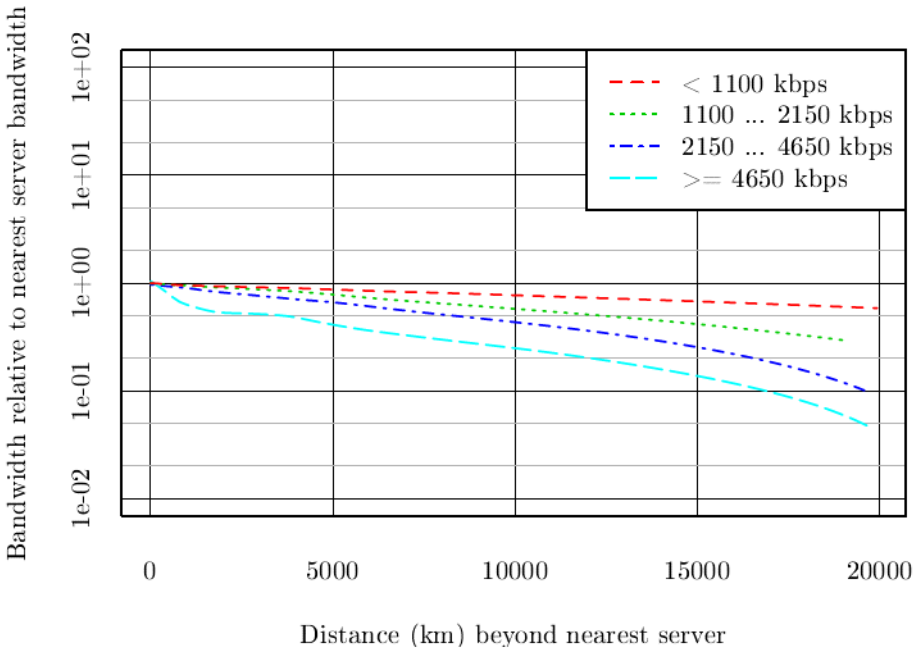


Figure 4.5.3: Bandwidth change subject to increased distance

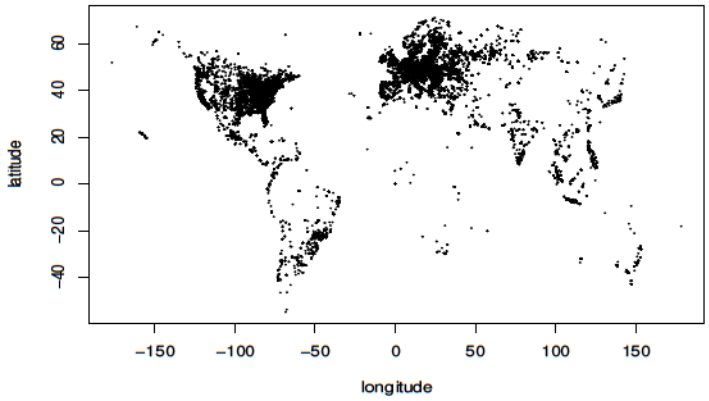


Figure 4.5.4: Host locations

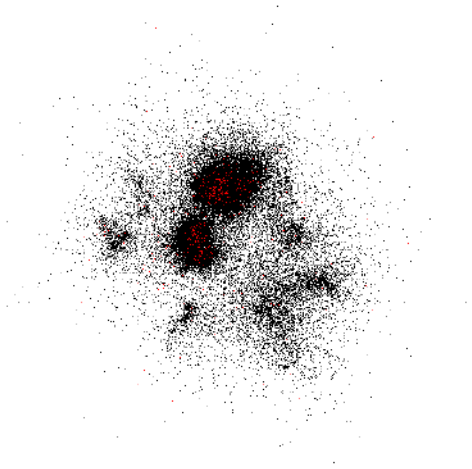


Figure 4.5.5: GNP delay space

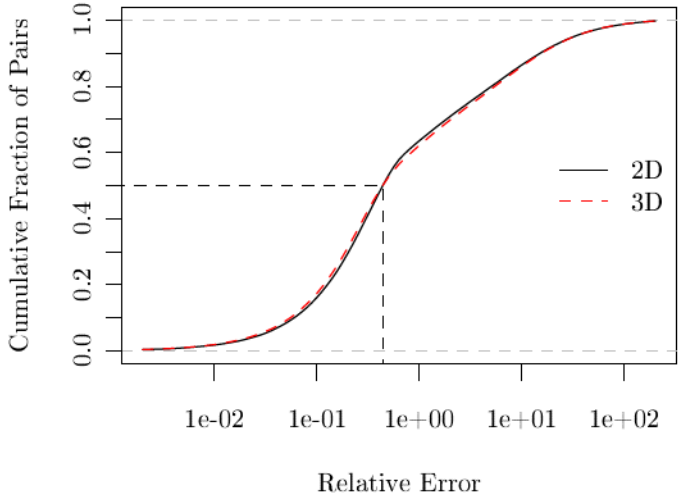


Figure 4.5.6: Cumulative error distribution

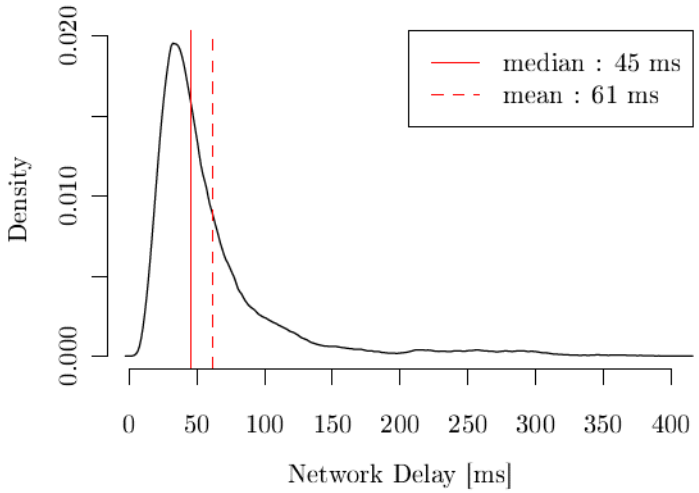


Figure 4.5.7: Histogram of hop delay distribution in Germany

Figure 4.5.3 shows for each user how the measured upstream bandwidth changes with server distance, relative to the closest server. LOWESS [35] regression curves are drawn for different maximum bandwidth ranges. For three-fourth of the users, average bandwidth only decreases slightly with increased connection length, i. e. their access link is indeed the bottleneck in their end-to-end Internet connections. In fact, this assumption seems to hold reasonably well for the domestic connections of 90 % of the users. Only for connections with upstream bandwidths larger than 4650 kbps the bandwidth drops significantly on the first 1000 km.

The locations of the nodes, according to MaxMind's GeoIP are shown in Figure 4.5.4.

After separating the delays as described in Section 4.5.2, we inserted the hosts into a 2-dimensional delay space, using their geographical locations as initial hint for the Vivaldi optimizer. After about 10,000 iterations the algorithm converged to the configuration shown in Figure 4.5.5. The red dots represent the reference servers, pulled in-place by their neighbors. The cumulative distribution of the relative error between the measured latencies and the final distance in delay space is shown in Figure 4.5.6. Interestingly, increasing the dimensionality of the delay space to three did not lead to a significant reduction of relative error, this is consistent with observations of [38]. Latency-wise, as it seems, the earth is still rather flat.

Figure 4.5.7 shows the resulting probability density estimate of end-to-end delay for packets of 400 bytes between arbitrary hosts in Germany. Although half of the delays are less than 45 ms, the long tail is clearly visible.

4.6 Conclusion

It was shown that end-to-end delay of packets passing through consumer Internet access links exhibits jitter and is very sensitive to congestion. This fact is disregarded by existing statistical and GNP models. Until now, its simulation required potentially complex topology models. The DELFOI delay simulator for OMNeT++ presents a hybrid approach that is already capable of simulating these end-to-end delays much more realistically than existing Internet delay models without topology modeling, yet requires only moderate resources.

End-to-end delay matrices are unfeasible to obtain for a non-trivial number of hosts. We have presented a reasonable parametrization for a global network delay model from co-operative HTTP delay measurements, exhibiting a broad range of network performance observations. The model converges even in the absence of landmarks, though the relative fitting errors are notedly higher than those in, e. g. [38]. Clearly, speedtest.net's data set exhibits much more heterogeneity, e. g. triangle inequality violations, than that of PlanetLab with mostly university hosts [44].

The modeling of the Internet access link could be further enhanced by taking, e. g. network interface queue length and delay caused by DSL interleaving and cable access control into account [45]. Optimally, the jitter model should be parametrized from the same data source as the others. The jitter data provided by the PingER Project consists of an average/expectation value and the inter-quartile-range (IQR). Currently these values are used to fit a log-normal distribution (4.4.2) that is used as a source for random jitter values. A more detailed jitter source, e. g. a fractal model that accounts for burstiness and correlation of jitter, would make these delay variations more realistic.

Currently, the server nodes are only "pushed" in place by connected client nodes. The use of a set of landmarks may speed up the convergence and improve overall accuracy.

Chapter 5

Emulating and Visualizing Periodic Communication

I hear and I forget; I see and I remember; I do and I understand.
Chinese Proverb[1]

Contents

Modeling, simulation as well as visualization are essential tools for interactive concept discovery, as well as researching and evaluating the behavior of complex distributed systems. However, the simulation of large scale peer-to-peer streaming applications, which send packets at high frequencies, is associated with a considerable computational effort, that makes simulation-based interactive exploration unfeasible.

By employing steady state emulation instead of discrete event simulation, resulting overall latencies can be computed directly as a function of transmission timing and network delay. In this chapter, I present REEL[2], a JAVA tool that facilitates the conceptual and operational modeling, emulation, visualization, and interactive exploration of communication in periodic peer-to-peer hypercube gossiping, especially its timing behavior

[1] after Confucian philosopher Xún Zǐ, ca. 310 BC-237 BC
[2] Reel-time Explorer Emulating Latencies

and latencies. The tool's interactive and experiment modes proved helpful in the analysis of the magnitude and composition of latency of the communication scheme presented in Section 3.4. Particularly, it helped me to study the timing interrelations of this overlay network, derive analytical latency bounds, perform simulations with realistic network delays, and assess latency minimization algorithms, presented in Chapter 6. Last but not least, it helped to verify the behavior of the implementation of periodic peer-to-peer hypercube gossiping, presented in Chapter 7.

5.1 Scientific Discovery Learning using Interactive Modeling

A system comprised of relatively simple parts may exhibit surprisingly complex behavior where small changes to its input may have grave effects on its output, according to some non-obvious pattern. Most fast moving animals, including humans, have developed a strong ability to quickly detect patterns visually [25]. As a consequence, the successful detection of patterns motivates the development of adequate visualization methodologies that make them easily perceptible. One application is the detection of patterns in output from pseudo random generators which in turn utilize chaotic models to hide their inevitable determinism [108, 128]. Figure 5.1.1 shows two random bitmaps, one created from the true random source RANDOM.ORG[3] that uses unpredictable physical measurements and one from the pseudo random generator behind PHP `rand()` function on Microsoft Windows. Simple visualization renders the deterministic pattern of the pseudo-random generator easily perceptible [2].

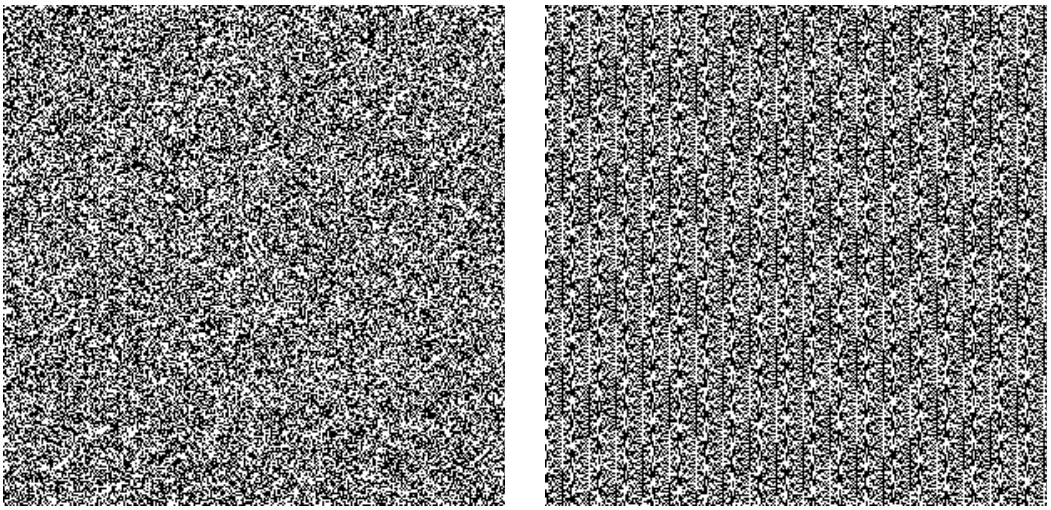


Figure 5.1.1: Random bitmaps created from random.org (left) and PHP rand() on Microsoft Windows (right) [2]. The intentionally complex but deterministic behavior of the PRNG is exposed by means of simple visualization.

In order to wield a complex system, one has to gain a profound understanding of its behavior. Knowing its elementary operations may be necessary but is certainly not suf-

[3] `http://www.random.org/`

ficient, just as knowing the rules by which to move the pieces in a board game is a long way from mastering the game; mastery usually requires practical experience best gained by either watching or, even better, playing the game. The essential task of finding and representing patterns and regularities and developing and verifying hypotheses about a system previously unknown is known as *concept discovery* [36]. In *scientific discovery learning* with simulations, the learner infers characteristics by changing input variables and observing the resulting changes in the values of the output variables [41]. Interactive models with three-dimensional visualizations greatly support conceptualization of abstract scientific phenomena: Noteworthy examples include the use of three-dimensional computer modeling ("Virtual Solar System") to deepen undergraduate students' understanding of astronomy concepts [79], visual molecular modeling in conjunction with multiple simultaneous representations in eleventh grade chemistry [150], or even the use of the computer game CIVILIZATION III to teach complex interplay of world history, geography, politics, and economics in select world history education classes [139].

These successful examples, however, describe the use of interactive modeling in education. The simulators were designed beforehand by or according to the instructions of persons already possessing profound knowledge of the domain; the learners were mere users, unable to modify the simulator. Here, by contrast, the simulator is developed by the learner (me) and refined in the course of subsequent knowledge acquisition.

The domain regarded in the following Chapter 6 comprises the influence of timing in periodic peer-to-peer hypercube communication on the resulting overall latency. Since real-time/live streaming applications require most current data, excessive latency is to be reduced as much as possible. As Chapter 6 will show, a considerable fraction of this latency is caused by data sojourning at intermediate overlay nodes. While the wait latency obviously depends on the time instants at which packets are sent by nodes of the overlay network, the specific interrelations are not obvious. To facilitate concept discovery, gain essential initial ideas, develop and verify hypotheses, and infer characteristics of these interrelations, an interactive exploratory simulation tool is needed. It should allow natural manipulation of the model and provide intuitive interpretation of simulation results in a way that they can be easily grasped.

5.2 Emulation

The effort associated with the simulation of large scale peer-to-peer streaming application is immense because every node spouts packets with high frequency. Consequently, for interactive exploration, it is necessary to emulate network behavior instead of simulating it in every detail. Once the network has completed configuration and parameter tuning, it reaches a steady state where, apart from minor fluctuations, faithfully produced by underlay network model, all communication repeats itself. Streaming applications employ jitter buffers that compute quantiles of the jitter distribution. Although the application may process incoming packets as soon as they arrive, the timing and latencies are fully determined by these quantiles, and thus, static. For the exploration and evaluation of timing behavior and latencies, it is hence sufficient to regard the delay quantiles and treat the repeating sequence of events as single steady state. This approach is utilized by the REEL tool: By employing steady state emulation instead of discrete event simulation, resulting network and wait latencies for all source/destination pairs, as well as overall latencies can be computed directly as a function of the transmission timing and network delay. This is done by tracing all data paths and accumulating latencies on the way.

5.3 Modeling

Models are differentiated into two kinds: *conceptual models* and *operational models*. Conceptual models hold principles, concepts and facts governing (constraining) the system, whereas operational models include sequences of operations (procedures) that can be applied to the system [41]. This concept is illustrated in Figure 5.3.1. The conceptual model

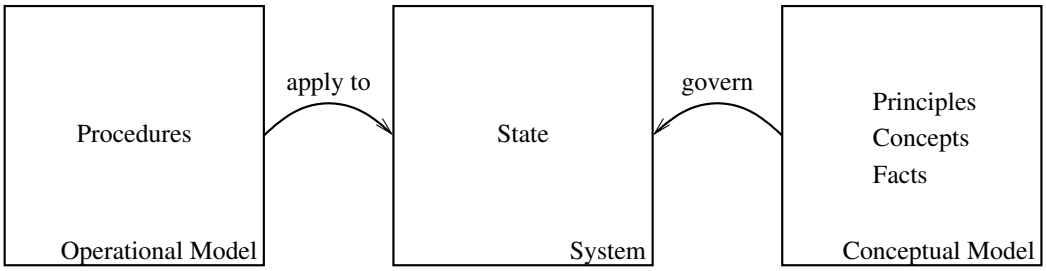


Figure 5.3.1: Model kinds

could, for instance, hold the laws of physics, and the operational model, the navigational control sequence of a space craft. However, the distinction is not in all cases an obvious one. Both the conceptual and the operational model narrow the state space of the system on the basis of rules. One could also consider the conceptual model as comprising the internal, system inherent rules and the operational model as comprising the external stimuli. An alternate view is to simply regard the conceptual model as fixed invariant and the operational model as replaceable strategy.

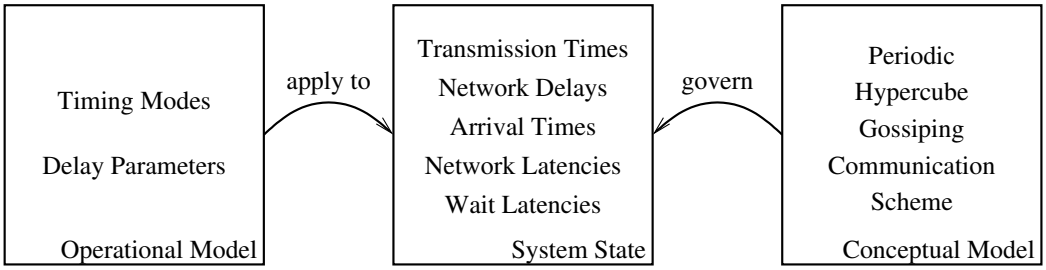


Figure 5.3.2: REEL models

Following this view, the REEL models are shown in Figure 5.3.2 and explained below:

System State

The system state of the REEL model holds the instants of time at which each node transmits a packet to a neighbor node, the network delays between all such neighboring nodes, the times at which the packets are received, and the overall network and wait latencies.

The system state contains no logic of its own. All values are determined by the operational and the conceptual model.

Conceptual Model

The conceptual model binds each arrival time to the sum of its corresponding precedent transmission time and network delay. It also binds network and wait latencies to the sums of the individual delays by tracing all network paths according to the periodic hypercube gossiping communication scheme (Section 3.4).

Operational Model

The operational model will, when applied to the system state, alter the nodes' transmission times and network delays.

Transmission times may be either simply uniformly randomized, synchronized, or manipulated by various algorithms, called *timing modes*, presented in Chapter 6. In the most basic timing mode, called *Random Phase Mode*, the transmissions occur at arbitrary times. All other timing modes constrain the timing in some way, thereby further reducing the freedom (*slack*) of the system state. For instance, *I/O Splicing Mode* constrains each node's transmission times to integer multiples of a slot time, thus turning the afore continuous model into a discrete one.

Also part of the operational model are the network delay models and parameters. These are used to configure the network delay variables of the system state from miscellaneous sources, such as various synthetic probability distributions (e. g. rectangular or truncated normal distributions) or PeerfactSim.KOM's [119] more realistic network delay model, described in Subsection 4.3.2. This facilitates the use of *Global Network Positioning* [115] to generate realistic delays between nodes belonging to a particular group such as a particular city, country or continental region. PeerfactSim's network delay model also supports inter-region-specific jitter and was in the course of this work specially extended to support node specific access delays. Besides the use of timing modes, that manipulate the transmission times in some particular, explicit way, REEL offers a domain model agnostic, simulated annealing [80] driven latency optimization feature. Before some of the interrelations were discovered and the advanced timing modes developed, this early feature proved especially helpful to compare favorable and unfavorable configurations and thereby generate ideas about the model, like well designed experiments can do in the lack of a hypothesis [41].

5.4 Visualization

To illustrate delays, I propose the use of *timed sequence graphs (TSGs)* as the one shown in Figure 5.4.1 and belonging to a communication scheme for 16 nodes, naïve transmission timing and constant network delays. Each (black) horizontal line represents one node. The abscissal arrow to the bottom indicates the flow of time. From the end points of the line segments between one node and another, the time instances at which data packets are sent and received can be observed. Thus, the horizontal component of each line segment represents the network hop delay for that particular sender/receiver pair. In the figure, the network delays are all the same, in general they are not. The widths of TSGs correspond to one period T. For illustration purposes, the send events are synchronized. The colors of the arrows represent the different levels of the communication scheme (Section 3.4) and are mapped as follows (from left to right): Ł$_0$:red, Ł$_1$:green, Ł$_2$:blue, Ł$_3$:yellow.

As indicated in Subsection 3.4.1, in streaming all communication recurs periodically and events are characterized by phases $\varphi \in D$. The domain $D = \{\mathbb{R}/0 \leq \varphi < T\}$ of φ can be

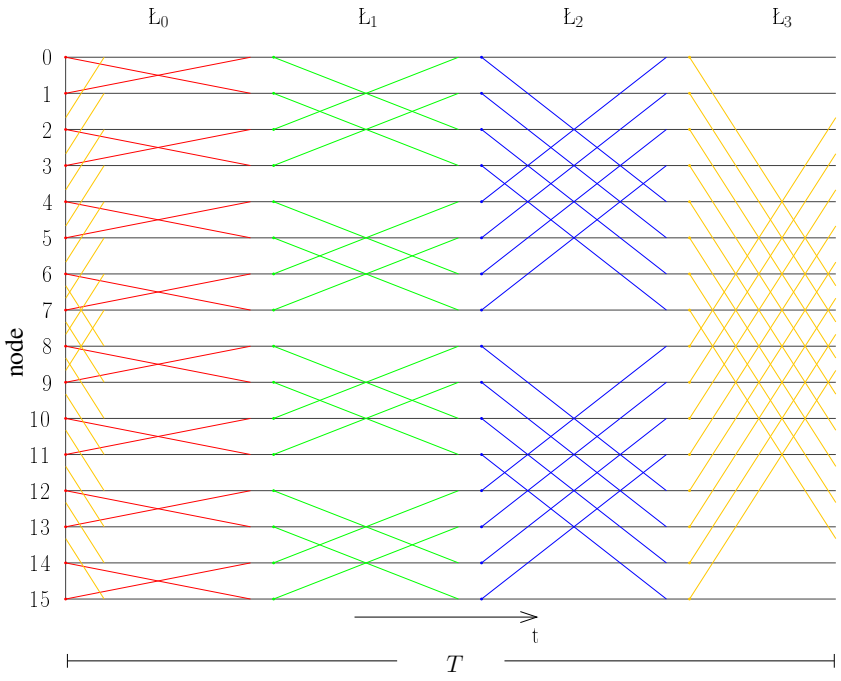


Figure 5.4.1: Timed sequence graph for a communication scheme of 16 nodes, trivial transmission timing, and constant network delays

viewed as a circle S^1 with radius T/τ with $\tau = 2\pi$. Accordingly, Figure 5.4.2b shows the timed sequence graph of Figure 5.4.1 in cylindrical representation.

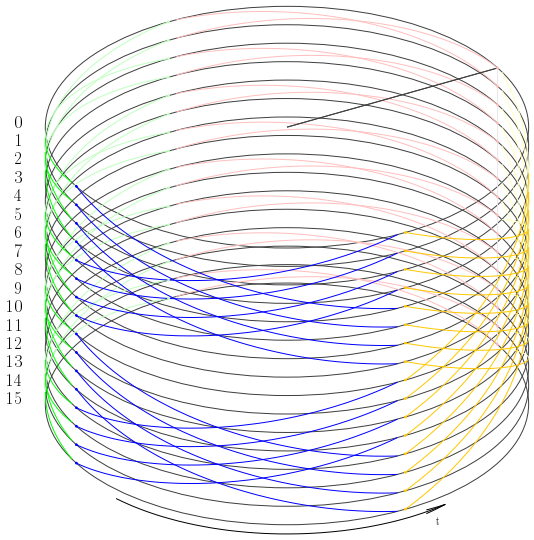


Figure 5.4.2: Timed sequence graph (cylindrical representation)

5.5 The REEL Tool

To facilitate running many simulations in an automated fashion and performing experiments, the REEL tool may be started from the command line in non-interactive mode, whereupon it performs simulations with varying parametrizations and writes simulation results for each iteration.

When launched in interactive mode, REEL shows a JAVA-SWING-based [140] graphical user interface *(main view)* produced in Figure 5.5.1. The main view is roughly divided into four columns, from left to right: Sequence view (the "reel"), Transmission phase parametrization, Network delay parametrization, and Results and statistics.

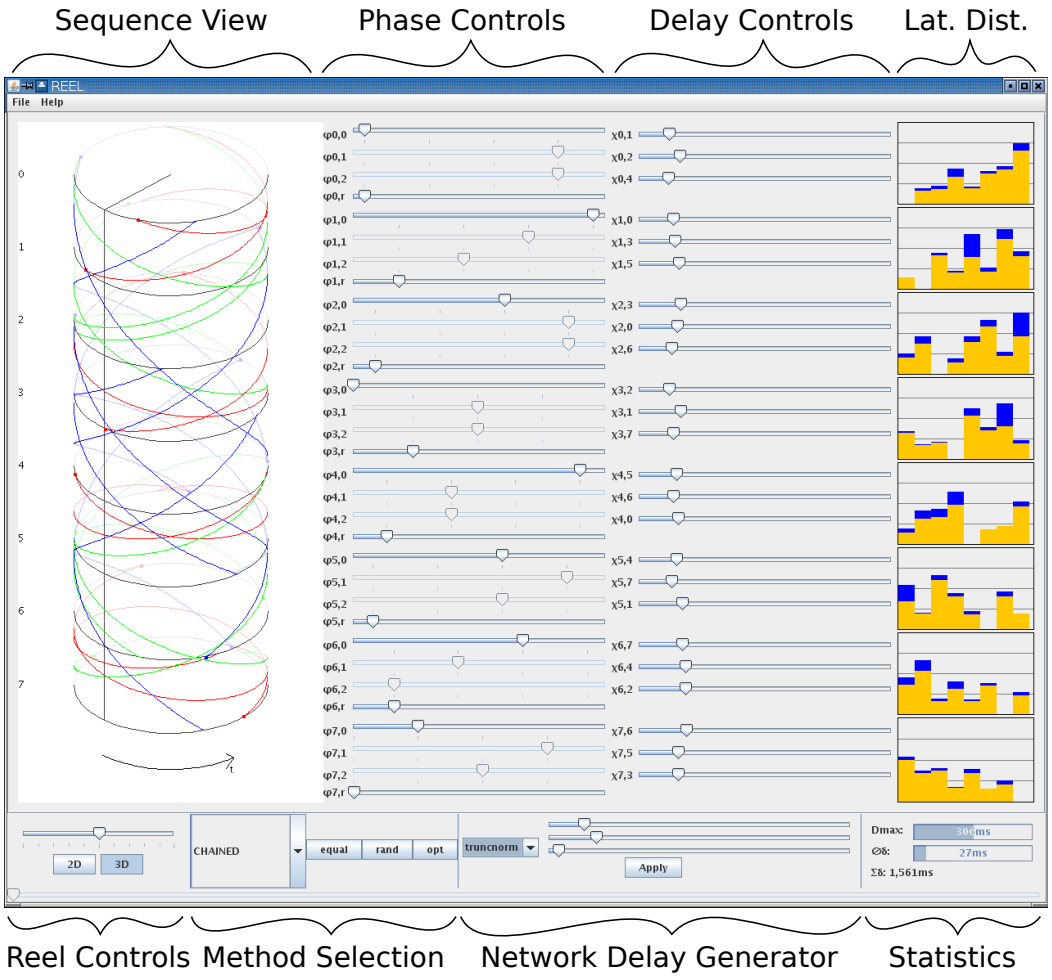


Figure 5.5.1: REEL main view

5.5.1 Sequence View

The sequence view visualizes the timing of the periodic peer-to-peer communication model, using timed sequence graphs, as described in the previous Section 5.4.

The reel controls at the bottom of REEL's main view drive the reel's rotation and allow to switch between 2-D (flat) and 3-D (cylindrical) view.

Changes to the model are reflected in reel-time[4]. An export function allows to save snapshots of the sequence view in the vector-based `xfig` format.

5.5.2 Phase Parametrization

The phase sliders allow to adjust each node's transmission times to each of its neighbors, as well as the I/O (e. g. record, playback) phases, i. e. the times at which a chunk of data becomes available at a node or can be consumed, respectively. Depending on the timing mode in effect, the choices are constrained. In *Source/Drain Sync Mode,* the source (e. g. record) event is used to trigger the sending of all transmissions. Consequently, the individual transmission phase sliders are disabled and follow the I/O phase sliders. In I/O Splicing Mode, transmissions are only possible at certain equidistant phases, their relative offsets being given by the I/O phase sliders. Consequently, in this discrete timing mode the phase sliders snap to the ticks, which are also shifted along, following the I/O sliders.

5.5.3 Network Delay Parametrization

The network delay sliders allow to parametrize network delay from each node to each of its neighbors. While configuring network delays individually is not feasible for most simulations, it allows, on the one hand, to interactively observe the overall effects of delaying just one transmission, and on the other hand, to fabricate pathological (e. g. worst-case) scenarios that can be very insightful in studying system behavior. The network delay generator controls at the bottom allow to sample from aforementioned synthetic distributions (rectangular or truncated normal distribution) or the GNP-based global delay model, all part of the operational model.

5.5.4 Results and Statistics

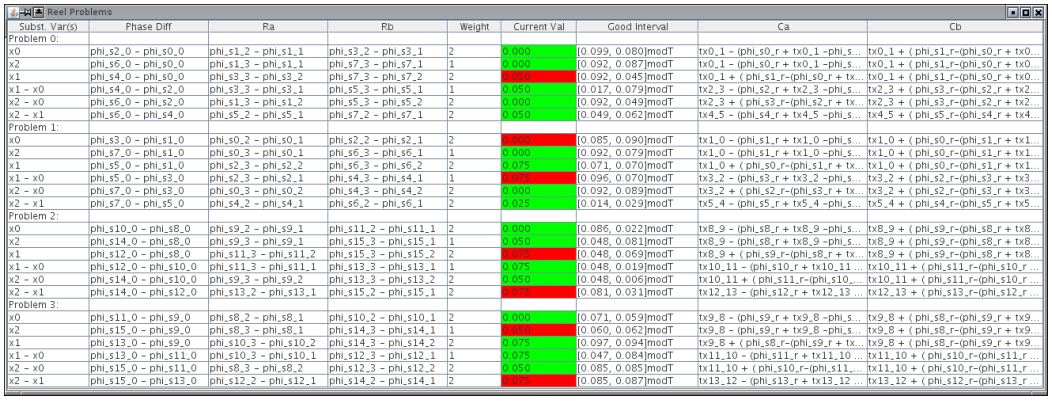
The rightmost column shows the simulation results, i. e. the distribution and composition of latency for all ordered pairs of nodes. Each node's bar chart shows by what amount each other node's piece of information is delayed, with the positions of the bars corresponding to the sources. The yellow bottom part of each bar represents network latency (i. e. sum of network delays) whereas the blue top portion represents wait latency (i. e. sum of wait delays) on the respective path from the particular source to the destination node in question. The bar charts allow for an instant, intuitive apprehension of resultant wait latency in relation to (more or less inevitable) network latency. The statistics display at the bottom provides information about the maximum, average and total latency.

5.5.5 Problems View

As will be particularized in Subsection 6.2.5, one particularly noteworthy discovery with regard to total wait latency was made in *Chained Mode* (Subsection 6.2.4), at which packets to higher level neighbors are sent promptly after the reception of packets from lower level neighbors: The sums of some wait delays were found to be periodic rectangular functions and the global latency minimum not a point but a plateau. Using the REEL tool, I was

[4] i. e. the next time the reel is drawn. If the reel is currently not rotating, a timely redraw is scheduled. To the human beholder, the change appears to happen instantaneously, i. e. in real-time.

able to further investigate this particularity and to formulate conditions for favorable sub-configurations. In a tabular problems view, produced in Figure 5.5.2, REEL displays the conditions for which each sum of complementary wait delays is minimal and whether these conditions are currently met.


Reel Problems

Subst. Var(s)	Phase Diff	Ra	Rb	Weight	Current Val	Good Interval	Ca	Cb
Problem 0:								
x0	phi_s2_0 – phi_s0_0	phi_s1_2 – phi_s1_1	phi_s3_2 – phi_s3_1	2	0.000	[0.099, 0.080]modT	tx0_1 – (phi_s0_r + tx0_1 –phi_s...	tx0_1 + (phi_s1_r–(phi_s0_r + tx0...
x2	phi_s6_0 – phi_s0_0	phi_s1_3 – phi_s1_1	phi_s7_3 – phi_s7_1	1	0.000	[0.092, 0.087]modT	tx0_1 – (phi_s0_r + tx0_1 –phi_s...	tx0_1 + (phi_s1_r–(phi_s0_r + tx0...
x1	phi_s4_0 – phi_s0_0	phi_s3_3 – phi_s3_2	phi_s7_3 – phi_s7_2	2	0.050	[0.092, 0.045]modT	tx0_1 + (phi_s1_r–(phi_s0_r + tx...	tx0_1 + (phi_s1_r–(phi_s0_r + tx0...
x1 – x0	phi_s4_0 – phi_s2_0	phi_s3_3 – phi_s3_1	phi_s5_3 – phi_s5_1	1	0.050	[0.017, 0.079]modT	tx2_3 – (phi_s2_r + tx2_3 –phi_s...	tx2_3 + (phi_s3_r–(phi_s2_r + tx2...
x2 – x0	phi_s6_0 – phi_s2_0	phi_s1_3 – phi_s1_2	phi_s5_3 – phi_s5_2	2	0.000	[0.092, 0.049]modT	tx2_3 + (phi_s3_r–(phi_s2_r + tx...	tx2_3 + (phi_s3_r–(phi_s2_r + tx2...
x2 – x1	phi_s6_0 – phi_s4_0	phi_s5_2 – phi_s5_1	phi_s7_2 – phi_s7_1	2	0.050	[0.049, 0.062]modT	tx4_5 – (phi_s4_r + tx4_5 –phi_s...	tx4_5 + (phi_s5_r–(phi_s4_r + tx4...
Problem 1:								
x0	phi_s3_0 – phi_s1_0	phi_s0_2 – phi_s0_1	phi_s2_2 – phi_s2_1	2	0.000	[0.085, 0.090]modT	tx1_0 – (phi_s1_r + tx1_0 –phi_s...	tx1_0 + (phi_s0_r–(phi_s1_r + tx1...
x2	phi_s7_0 – phi_s1_0	phi_s0_3 – phi_s0_1	phi_s6_3 – phi_s6_1	1	0.000	[0.092, 0.079]modT	tx1_0 – (phi_s1_r + tx1_0 –phi_s...	tx1_0 + (phi_s0_r–(phi_s1_r + tx1...
x1	phi_s5_0 – phi_s1_0	phi_s2_3 – phi_s2_2	phi_s6_3 – phi_s6_2	2	0.075	[0.071, 0.070]modT	tx1_0 + (phi_s0_r–(phi_s1_r + tx...	tx1_0 + (phi_s0_r–(phi_s1_r + tx1...
x1 – x0	phi_s5_0 – phi_s3_0	phi_s2_3 – phi_s2_1	phi_s4_3 – phi_s4_1	1	0.075	[0.096, 0.070]modT	tx3_2 – (phi_s3_r + tx3_2 –phi_s...	tx3_2 + (phi_s2_r–(phi_s3_r + tx3...
x2 – x0	phi_s7_0 – phi_s3_0	phi_s0_3 – phi_s0_2	phi_s4_3 – phi_s4_2	2	0.000	[0.092, 0.089]modT	tx3_2 + (phi_s2_r–(phi_s3_r + tx...	tx3_2 + (phi_s2_r–(phi_s3_r + tx3...
x2 – x1	phi_s7_0 – phi_s5_0	phi_s4_2 – phi_s4_1	phi_s6_2 – phi_s6_1	2	0.025	[0.014, 0.029]modT	tx5_4 – (phi_s5_r + tx5_4 –phi_s...	tx5_4 + (phi_s4_r–(phi_s5_r + tx5...
Problem 2:								
x0	phi_s10_0 – phi_s8_0	phi_s9_2 – phi_s9_1	phi_s11_2 – phi_s11_1	2	0.000	[0.086, 0.022]modT	tx8_9 – (phi_s8_r + tx8_9 –phi_s...	tx8_9 + (phi_s9_r–(phi_s8_r + tx8...
x2	phi_s14_0 – phi_s8_0	phi_s9_3 – phi_s9_1	phi_s15_3 – phi_s15_1	1	0.050	[0.048, 0.081]modT	tx8_9 – (phi_s8_r + tx8_9 –phi_s...	tx8_9 + (phi_s9_r–(phi_s8_r + tx8...
x1	phi_s12_0 – phi_s8_0	phi_s11_3 – phi_s11_2	phi_s15_3 – phi_s15_2	2	0.075	[0.048, 0.069]modT	tx8_9 + (phi_s9_r–(phi_s8_r + tx...	tx8_9 + (phi_s9_r–(phi_s8_r + tx8...
x1 – x0	phi_s12_0 – phi_s10_0	phi_s11_3 – phi_s11_1	phi_s13_3 – phi_s13_1	1	0.075	[0.048, 0.019]modT	tx10_11 – (phi_s10_r + tx10_11 ...	tx10_11 + (phi_s11_r–(phi_s10_r ...
x2 – x0	phi_s14_0 – phi_s10_0	phi_s9_3 – phi_s9_2	phi_s13_3 – phi_s13_2	2	0.050	[0.048, 0.006]modT	tx10_11 + (phi_s11_r–(phi_s10_...	tx10_11 + (phi_s11_r–(phi_s10_r ...
x2 – x1	phi_s14_0 – phi_s12_0	phi_s13_2 – phi_s13_1	phi_s15_2 – phi_s15_1	2	0.075	[0.081, 0.031]modT	tx12_13 – (phi_s12_r + tx12_13 ...	tx12_13 + (phi_s13_r–(phi_s12_r ...
Problem 3:								
x0	phi_s11_0 – phi_s9_0	phi_s8_2 – phi_s8_1	phi_s10_2 – phi_s10_1	2	0.000	[0.071, 0.059]modT	tx9_8 – (phi_s9_r + tx9_8 –phi_s...	tx9_8 + (phi_s8_r–(phi_s9_r + tx9...
x2	phi_s15_0 – phi_s9_0	phi_s8_3 – phi_s8_1	phi_s14_3 – phi_s14_1	1	0.050	[0.060, 0.062]modT	tx9_8 – (phi_s9_r + tx9_8 –phi_s...	tx9_8 + (phi_s8_r–(phi_s9_r + tx9...
x1	phi_s13_0 – phi_s9_0	phi_s10_3 – phi_s10_2	phi_s14_3 – phi_s14_2	2	0.075	[0.097, 0.094]modT	tx9_8 + (phi_s8_r–(phi_s9_r + tx...	tx9_8 + (phi_s8_r–(phi_s9_r + tx9...
x1 – x0	phi_s13_0 – phi_s11_0	phi_s10_3 – phi_s10_1	phi_s12_3 – phi_s12_1	1	0.075	[0.047, 0.084]modT	tx11_10 – (phi_s11_r + tx11_10 ...	tx11_10 + (phi_s10_r–(phi_s11_r ...
x2 – x0	phi_s15_0 – phi_s11_0	phi_s8_3 – phi_s8_2	phi_s12_3 – phi_s12_2	2	0.050	[0.085, 0.085]modT	tx11_10 + (phi_s10_r–(phi_s11_...	tx11_10 + (phi_s10_r–(phi_s11_r ...
x2 – x1	phi_s15_0 – phi_s13_0	phi_s12_2 – phi_s12_1	phi_s14_2 – phi_s14_1	2	0.075	[0.085, 0.087]modT	tx13_12 – (phi_s13_r + tx13_12 ...	tx13_12 + (phi_s12_r–(phi_s13_r ...

Figure 5.5.2: Problems view

5.5.6 Geometrical Solution View

For four-dimensional hypercube (4-cube) based networks, the minimum latency solution set to each of the sub-problems, described in previous Subsection 5.5.5, is rendered as an animated graphical projection and displayed in the geometrical problem views, produced in Figure 5.5.3. The red polytope demarks the solution set, i. e. the set of points in parameter space for which the sum of latencies corresponding to the sub-problem is minimal. The mathematical derivation of this solution is expatiated in Subsection 6.2.5. A green sphere is inscribed into the largest polytope where multiple disjunct solutions exist.

The problems view and the solution view basically show the same information albeit in different representations, following the ideas of [84], who proposes the use of multiple linked representations of chemical experiments in teaching to effect deep conceptual understanding that exceeds superficial understanding gained from observing individual surface features alone.

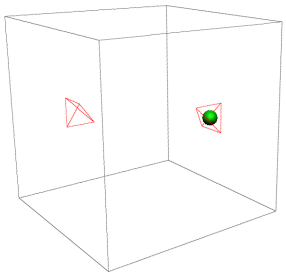


Figure 5.5.3: Geometrical solution view

5.5.7 Visualizing Real Application Measurements

In addition to visualizing simulation results from the REEL model, the tool also allows to load measured timing data from external simulators or real application runs. While the first constitute a static model, the latter are characterized by slight changes between communications rounds due to inaccurate or dynamically modified timing and fluctuations in network delay. Using the long slider at the very bottom of the main view, it is possible to reel through the recorded data in order to retrace log files and analyze system state properties and invariants.

5.6 Summary

The use of interactive modeling facilitates successful knowledge discovery, not only for students in well-established areas but also on new terrain. Using easy programming languages such as JAVA enables quick prototyping and specifically to adjust and extend the modeling tool on demand as new domain knowledge is acquired.

While the cylindrical sequence view best conveys the modulo concept of the periodic communication it proved less practical than the flat view in day to day use. This corresponds to the common use of terrestrial globes vs. maps.

The ability to use the same established representation for replaying measurements from application runs recorded in log files helped to bridge the usual gap between analytical modeling(“theory”) and simulation (“practice”).

Chapter 6

Minimizing Latency in Periodic Hypercube Gossiping

Contents

Applications of periodic hypercube gossiping range from scalable audio communication for massively multi-user virtual environments to decentralized data fusion for live object tracking to real-time business intelligence to shared haptic virtual environments. Despite their diversity, these applications share a common trait, namely their demand for the very latest data.

In most live streaming applications, latency reduces the value of received data or degrades overall service quality[1]. Consequently, it should be reduced as far as possible.

Moreover, in such real-time applications, not only the reduction of latency but also the ability to estimate and bound latency is of vital importance. In this context, worst-case traversal time (wctt) denotes the total time needed to transport a piece of information across the overlay network to its destination, including all network, wait and processing

[1] In audio conferencing, for instance, excessive latency disrupts conversation, as portrayed in Subsection 8.1.1. In live object tracking, regarded in Chapter 9, latency vitiates the value of state estimates.

delays. In periodic gossiping, it depends on the abstract communication complexity of the network topology, the physical properties of the underlay network, and the sojourn time of information at intermediate nodes.

Although gossiping belongs to the most investigated communication problems [9], prevalent complexity measures [27, 82, 66, 96] only consider one-shot dissemination, assume different communication constraints, or just regard abstract complexity, i. e. an integer number of communication rounds. Beyond that, communication schemes are downright latency agnostic and disregard diverse network delay or wait latency. Moreover, to my best knowledge, no time bounds for periodic hypercube gossiping have been published.

In this chapter, I present an analysis of latency for periodic gossiping in F1 mode on a regular binary hypercube overlay network. As will be shown, due to wait delay caused by data sojourning at intermediate overlay nodes, latency is highly dependent on the timing or synchronization of the communication, and thus, to considerable extent under direct control by the application. I will present five different disciplines governing node behavior, referred to as *timing modes*, and demonstrate how networks operating in different timing modes—though obeying the same communication scheme—bear considerably dissimilar traversal times.

Aforementioned abstract communication complexity will be complemented by a more detailed calculus and time bound that takes into account how long information is delayed by network- and wait-delays and derive analytical expressions for worst case and expected wait latency as well as multi-hop traversal time distributions. Finally, these analytical solutions will be corroborated with results from a steady state emulation using the REEL tool of Chapter 5 in conjunction with the GNP underlay network model, described in Chapter 4.

The rest of this chapter is structured as follows: Section 6.1 extends the hypercube gossiping nomenclature of Section 3.1 by variables and latency measures required for evaluating the performance of the subsequent timing modes. Section 6.2 presents five timing modes along with worst-case and expected wait latency as well as traversal time bounds. Section 6.3 compares the analytical solutions to results obtained by steady state emulation whereas Section 6.4 summarizes the findings and concludes this chapter.

For the theoretical foundations on gossiping, communication modes, and hypercubes refer to Chapter 2. The general hypercube gossiping communication scheme is introduced in Chapter 3. The analysis and evaluation was greatly aided by the REEL tool, portrayed in Chapter 5. Assumptions regarding network delay and an accordant model used for the simulations were laid out in Chapter 4.

6.1 Foundations

6.1.1 Latency Sources

In Subsection 3.5 various sources of latency are pointed out. In general, the two most significant ones are network delay and wait delay:

Network delay is the time, from the commencement of the sending of a data packet to the complete reception at a different node.

Wait delay is the time data sojourns at one node before it is used or passed on to another.

Delay Related Variables

Since both reception and transmission events take place periodically, each intermediate node or *"relay"* in the data path contributes up to one period T of wait delay. The delays depend on the offset of the send times of the nodes involved.

In addition to the nomenclature of Section 3.1, we define the following delay related variables, using the modulo notation introduced in Subsection 3.4.1:

$\varphi_{n,\ell}$ the phase (time instant within the period) when node n **sends** a packet to its neighbor on $Ł_\ell$

χ_{n_0,n_1} the time (single hop underlay network delay) it takes to transfer a packet from node n_0 to node n_1

$\theta_{n,\ell}$ the phase when node n **receives** a packet from its neighbor on $Ł_\ell$ with

$$\theta_{\mathrm{nb}(n,\ell),\ell} = \varphi_{n,\ell} \underset{T}{\oplus} \chi_{n,\mathrm{nb}(n,\ell)}$$

$\delta_{n,r,\ell}$ the wait delay, i. e. the time data sojourns or *ages* in node n after having been received from n's neighbor on $Ł_\ell$ and before being sent to n's neighbor on $Ł_r$

$$\delta_{n,r,\ell} = \varphi_{n,r} \underset{T}{\ominus} \theta_{n,\ell}$$

$ᴍ$ traversal time, i. e. the total time it takes information originating from a source node to reach a destination node, including network latency and wait latency.

k divisor of period T in Spliced and subsequent modes.

T_{slot} data procurement slot size $T_{\mathrm{slot}} := {}^{T}\!/_{k}$, i. e. minimum time interval between successively retrieving samples from the sensor.

ω random variable uniformly distributed between 0 and T_{slot}

Ω random variable uniformly distributed between 0 and T

Network Latency

Single-Hop Network Delay We define *single overlay hop network delay*, short *hop delay*, as the time span between the commencement of sending of a packet by one node and the instant it becomes fully available at its neighbor node. It is particularized in Section 4.2. The bold curve in Figure 6.1.1 shows a probability density estimate of minimal end-to-end delay for packets of 400 bytes between arbitrary hosts in Germany, March 2010. The estimate was obtained using the DELFOI model, developed in Section 4.4 and parametrized with measurement data obtained from `speedtest.net` as laid out in Section 4.5.

Since the distribution of Internet delay is long tailed [64], worst case hop delay may be considered infinite, tantamount to the loss of the packet. Instead of fixed worst-case times, live streaming applications employ timeouts that optimally conform to quantiles characterizing the time by which a packet will have arrived with a high probability. The choice of this quantile is a trade-off between the percentage of accommodated packets and the additional latency caused by jitter buffers. In latency critical applications, processing the majority of packets with low latency may be of higher value than accommodating for

rare packet delays of several seconds. The bold curve in Figure 6.1.2 depicts the quantile function of single hop delay.

Important, from an application's perspective, is the time by which a packet will have arrived with a high probability. $\chi_{n_1,n_2,q}$ denotes the time by which a (fixed-sized) packet of samples sent from node n_1 will have arrived at n_2 with probability q.

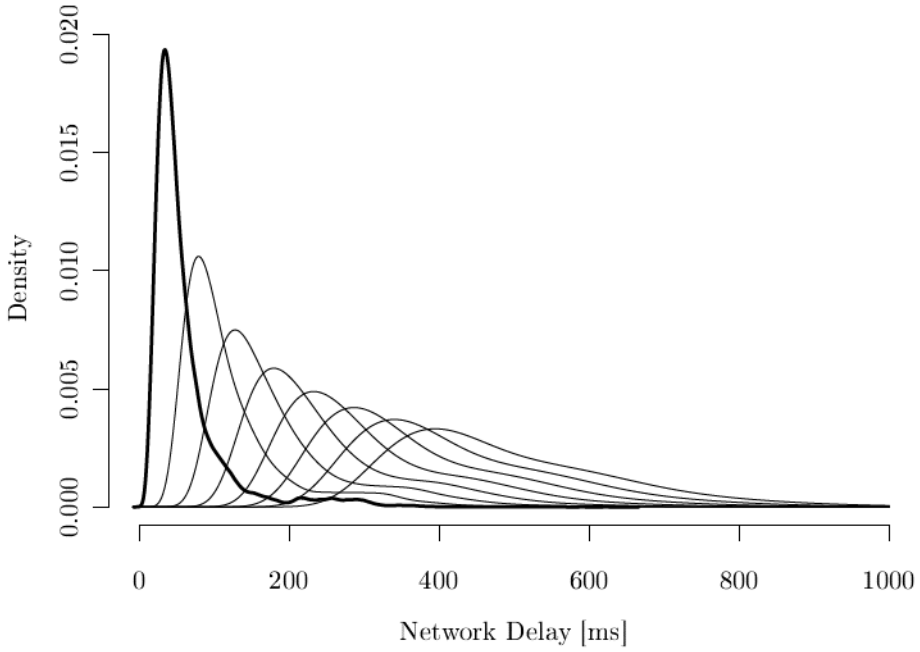


Figure 6.1.1: Probability density of (multi) hop delay for 400-byte packets in Germany. Bold curve: single hop delay

Multi-Hop Network Latency For multi-hop overlay paths, network latency $\overleftrightarrow{\chi}$ comprises the sum of the individual hop delays. Provided the individual hop delays are statistically independent, the distribution of the total path latency can be computed by the convolution of their distributions[2]. Further assuming a common hop delay distribution χ for all host pairs, the distribution of network latency for a path of L hops equals the L^{th} convolution power of χ. That is, the probable bound $\overleftrightarrow{\chi}_{L,q}$ denoting the latency value that, with probability q, will not be exceeded by the sum of L hop delays is given by

$$P\left(\sum_{h=1}^{L}\chi_h \le \overleftrightarrow{\chi}_{L,q}\right) \ge q \;\Leftrightarrow\; \overleftrightarrow{\chi}_{L,q} = \int_{-\infty}^{q} \chi^{*L}(x)dx \tag{6.1.1}$$

where χ^{*L} denotes the L-fold convolution of χ's probability density function (PDF). The thinner curves of Figure 6.1.1 show convolution powers 2 to 8. Note how for each additional hop, latency not only increases but becomes less predictable. Figure 6.1.2 depicts $\overleftrightarrow{\chi}_{L,q}$ for $L \in [1..8]$.

[2] Depending on the scenario, statistical independence may be a simplifying assumption. The individual hop delays could be correlated both along the path due to network locality and over time due to the congestion state of nodes.

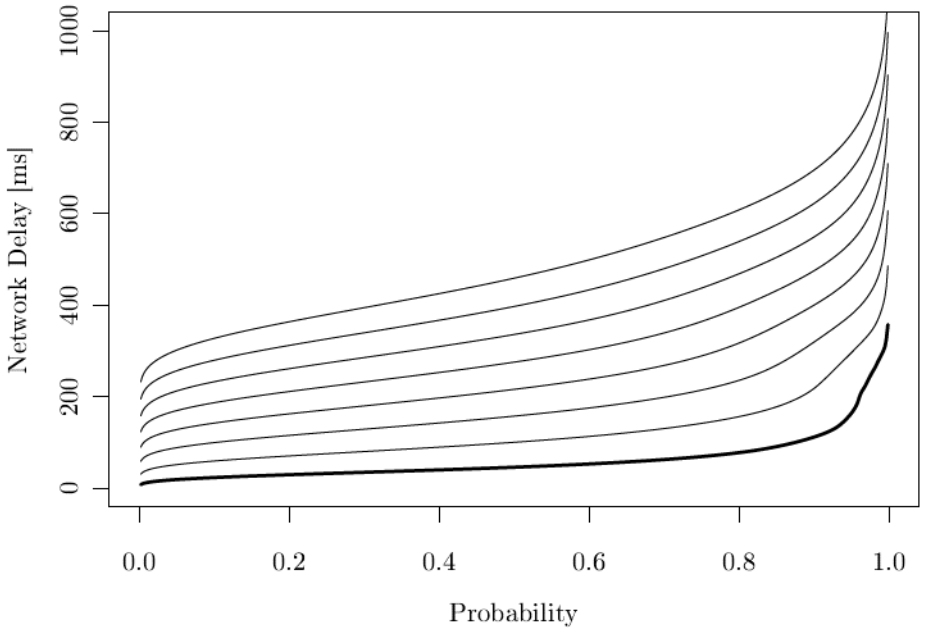


Figure 6.1.2: Quantile function of (multi) hop delay for 400-byte packets in Germany. Bold curve: single hop delay

For a hypercube network of L levels, the expected network latency follows from the hop-count distribution (3.3.1):

$$E[\overleftrightarrow{\chi}](L) = \frac{\sum_{h=1}^{L} \binom{L}{h} h}{2^L - 1} \overline{\chi} = L\overline{\chi} \tag{6.1.2}$$

Locality Some peer-to-peer networks can utilize the fact that hop delay between some nodes is less than between others to reduce latency. This exploitation of locality is known as *Proximity Neighbor Selection (PNS):* When a node has multiple choices of selecting its neighbors, it selects those that are closer in terms of latency [129]. For some applications (such as the one portrayed in Chapter 8), the choice of neighbors is determined by other constraints and PNS is not an option. Accordingly, in this chapter the node numbering and resulting network delays are being regarded as given, whereas the focus is on the reduction of wait latency. Still, network latency is accounted for in the estimation of traversal times, as is locality in the underlay network model used in the simulations.

Wait Latency

Wait Delay Wait delay δ denotes the time data sojourns at one node before it is passed on to another.

Lemma 6.1.1. *In the absence of inter-node-synchronization, δ is uniformly distributed between 0 and one period:*

Proof. As the arrival and send phases θ resp. φ are uniformly distributed, so is their difference δ as shown in Lemma 3.4.2. □

Wait Latency The sum of these delays (e. g. along an overlay path) is called wait latency.

I/O Levels Beside wait delay in between the receiving and the sending, data also sojourns after having been measured (and before being sent) and before being used (after having been received). By treating input and output as additional neighbors, these delays can be treated along with the other wait delays. Regard Figure 3.4.3 on page 45, showing a hypercube gossip communication scheme with L levels. Pertaining to wait latency, the same scheme plus an input and an output stage can be viewed as a scheme with $L + 2$ levels, as shown in Figure 6.1.3. The depicted scheme is used for audio conferencing, and

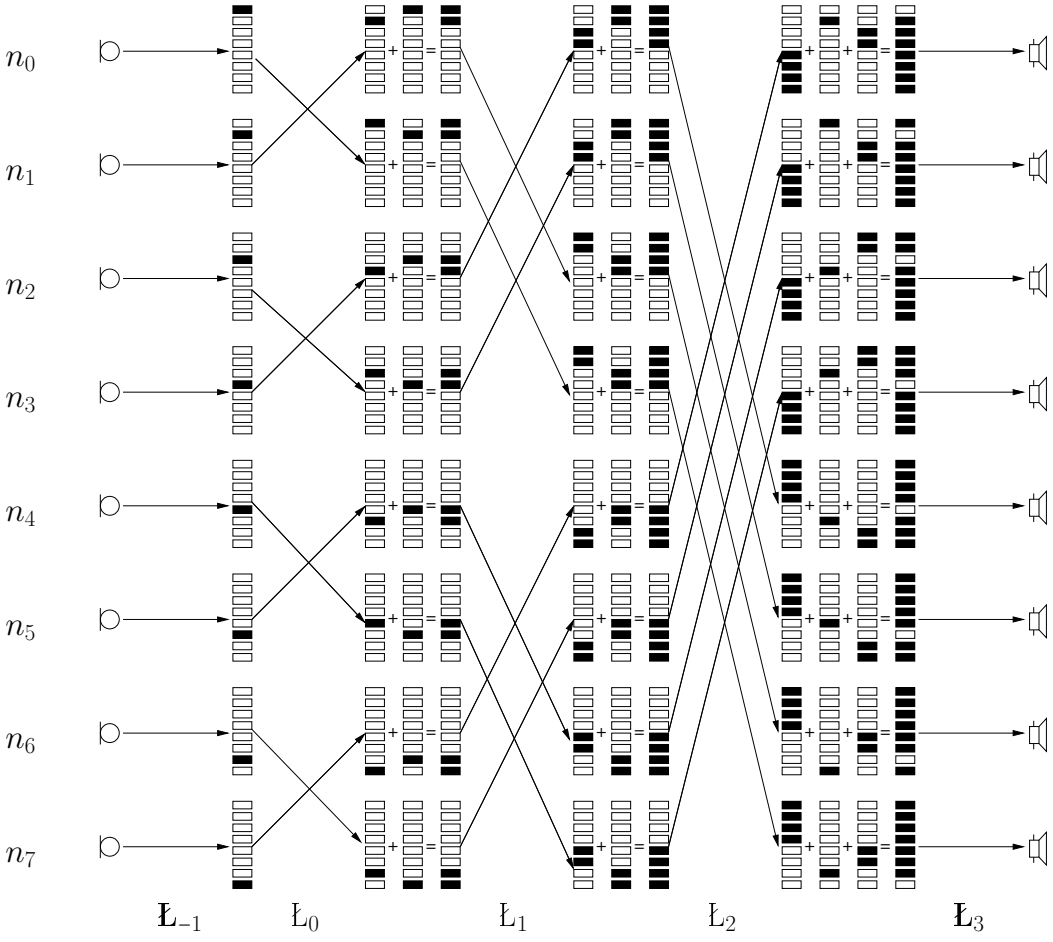


Figure 6.1.3: Hypercube gossip communication scheme for $b = 2$, $m = 3$ with extra levels Ł$_{-1}$ and Ł$_L$

thus, the final result does not contain each node's own source. In the remainder of this chapter, the input and output related virtual network levels will be denoted Ł$_{-1}$ and Ł$_L$, respectively. Accordingly, $\theta_{n,-1}$ denotes the sampling data procurement phase and $\varphi_{n,L}$, the data output phase.

Paths For the construction of latency distribution and traversal time quantiles we also need to consider individual source/destination combinations, or *paths*, and the frequencies of certain path lengths. Starting from an arbitrary node, the data paths to all other nodes may be represented as binary sequences of length L whereat the ℓ^{th} digit corresponds to the Ł$_\ell$ link, that can be either taken ("1") or not taken ("0"). For an example refer to Figure 6.1.4. Starting from node 6, the path %1010 (read from right to left) will contain

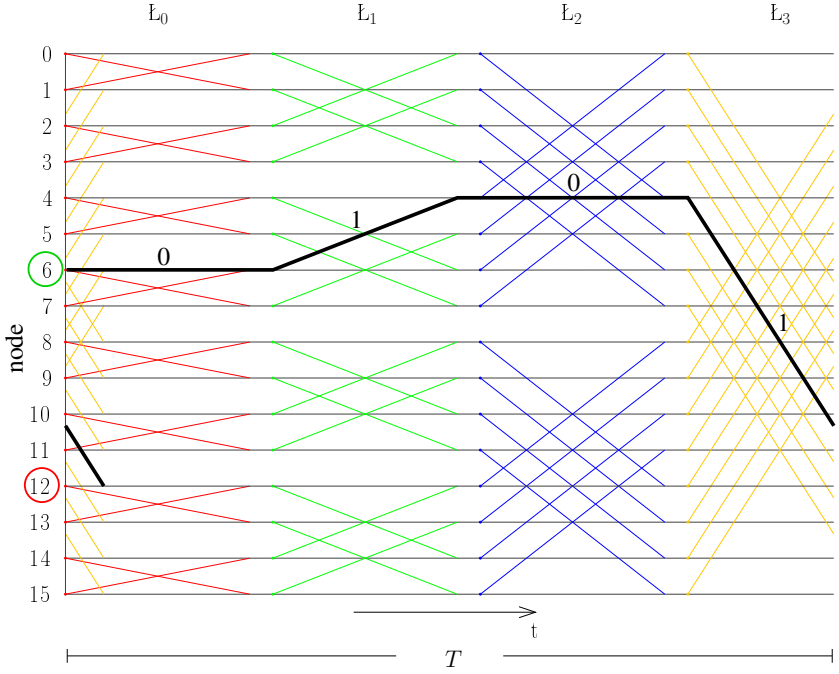


Figure 6.1.4: Timed sequence graph with path $(6, \%1010)$, from node 6 to node 12, highlighted

the following links: (6→7 not taken), 6→4, (4→0 not taken), 4→12, and will thus lead to node 12.

6.1.2 Latency Measures

Path Wait Latency

We let the *path wait latency* $\Delta_{i,j}$ denote the wait latency for source j observed at node i.

Least Upper Bound on Wait Latency

The wait latency bound $\hat{\Delta} := \sup_{\varphi,\chi,i,j} (\Delta_{i,j})$ denotes the wait latency that will not be exceeded for all parametrizations of φ and χ and all host/destination pairs. Note, however, that $\hat{\Delta}$ may, in some cases, not contribute to worst-case traversal time if hop delay exceeds wait delay, as shown below.

Overall Wait Latency

We let $\breve{\Delta}$ denote the overall wait latency, i. e. the total sum of all wait delays experienced by all nodes. Just as $\Delta_{i,j}$, it depends on the parametrizations of φ and χ.

$$\breve{\Delta} = \sum_{i=0}^{N-1} \sum_{j=0, j\neq i}^{N-1} \Delta_{i,j}$$

Mean Wait Latency

Each node receives information from all other nodes via overlay paths of different lengths according to the hop count distribution (3.3.1), consequently incurring different wait latencies. $\overline{\Delta_i}$ denotes the *mean wait latency* of all sources as experienced by node i:

$$\overline{\Delta}_i := \frac{\sum_{j=0, j\neq i}^{N-1} \Delta_{i,j}}{(N-1)}$$

The mean wait latency for all nodes of the network is defined as

$$\overline{\Delta} := \frac{\breve{\Delta}}{N \cdot (N-1)}$$

Least Upper Bound on Mean Wait Latency

The least upper bound on mean wait latency is defined as

$$\sup\left(\overline{\Delta}\right) = \sup_{\varphi,\chi,i}\left(\overline{\Delta}_i\right)$$

It is not so much used as a latency measure of its own but rather as a vehicle for proving expected wait latency, defined below.

Expected Wait Latency

In conjunction with $\hat{\Delta}$, the expectation $E[\Delta] = E\left[\overline{\Delta}\right]$ of wait latency is the most significant moment for the evaluation of the timing modes below.

Traversal Time

Traversal time denotes the total time it takes information originating from a source node to reach a destination node. Particularly, it comprises both network latency and wait latency. Of practical importance is the construction of some quantile of traversal time. Being the sum of individual delays, the probability distribution of traversal time can be computed from the convolution of the components' PDFs, their availability provided. The PDF of the overall traversal time distribution is the arithmetic mean of the individual paths' PDFs. In the equations below, all occurrences of the random variables ω, Ω, χ, and M are to be understood as to denote their respective PDFs.

6.2 Timing Modes

In this section we embrace the reduction of wait latency, which is inherent to periodic gossiping and to considerable extent under direct control by the application. Accordingly, we will develop several timing modes and compare them with regard to resulting latency. Unless noted otherwise we assume all φ, $\chi \bmod T$, and hence, θ to be uniformly distributed between 0 and T.

6.2.1 Random Mode

We first consider the general case, i. e. the implementation of the communication scheme as given in Section 3.4 without further constraints, called *Random Mode*. Figure 6.2.1 shows the timing of periodic gossiping on a hypercube network of eight nodes in Random Mode. Ł$_{-1}$ and Ł$_L$ are not shown. All transmissions are completely independent from each other.

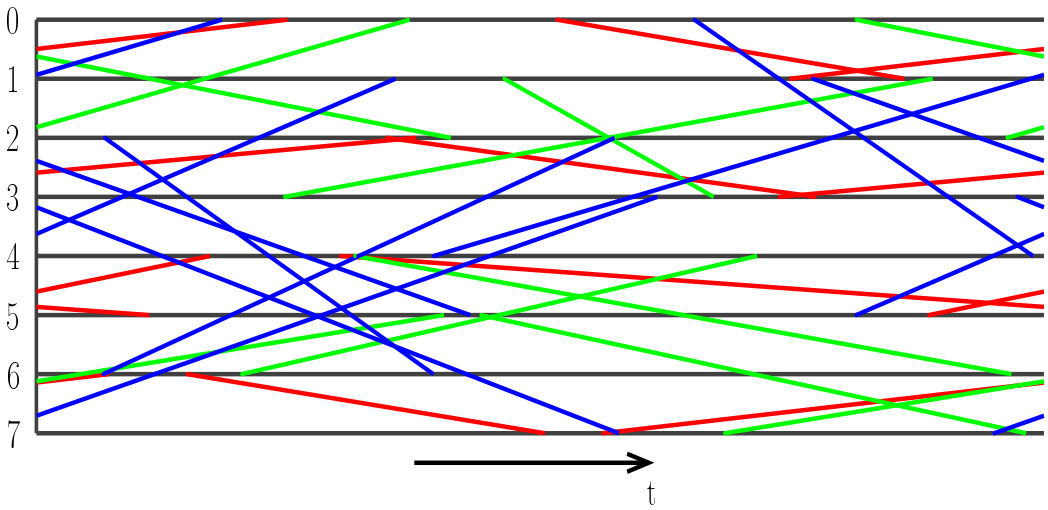


Figure 6.2.1: TSG for H_3 in Random Mode

$\hat{\Delta}_{\text{Random}}$ denotes the supremum of wait latency in Random Mode.

Theorem 6.2.1.

$$\hat{\Delta}_{\text{Random}} = (L+1)T.$$

Proof. The network considered has $L+2$ levels (network+I/O), resulting in $L+1$ (inter-level) wait delays, each of which contributing up to one period T of wait delay and thus rendering the maximum wait latency.

In other words, every node both receives and sends data packets at arbitrary times. Also data procurement and data output happen at random phases. Every node receives data from the most distant node via L overlay hops, experiencing wait delay $L-1$ times. Because the first transmission happens at an arbitrary phase after measurement and data usage happens at an also arbitrary phase after the last reception, each packet is additionally delayed at the input and at the output stage, accounting for two additional wait delays. Each wait delay is of length T at worst. □

Lemma 6.2.2. *The least upper bound on mean latency in Random Mode is given by*

$$\sup\left(\overline{\Delta}_{\text{Random}}\right) = \left(2^{L-1}L + 2^L - 1\right)\frac{T}{2^L - 1}$$

Proof. We assume the worst-case parametrization of φ and χ (i. e. wait delays of T at every possible position) and average over all paths:

$$\begin{aligned}
\sup\left(\overline{\Delta}_{\text{Random}}\right) &= \frac{\sup_{\varphi,\chi}\left(\sum_{j=0,j\neq i}^{2^L-1}\Delta_{i,j}\right)}{2^L-1} = \frac{\left(\sum_{h=1}^{L}\left(c_2(L,h)\cdot(h-1)\right) + 2\left(2^L-1\right)\right)T}{2^L-1} \\
&= \left(\sum_{h=1}^{L}\left(\binom{L}{h}\cdot(h-1)\right) + 2\left(2^L-1\right)\right)\frac{T}{2^L-1} \\
&= \left(\left(\left(\frac{1}{2}L-1\right)2^L+1\right) + 2\left(2^L-1\right)\right)\frac{T}{2^L-1} \\
&= \left(\left(\frac{1}{2}L+1\right)2^L-1\right)\frac{T}{2^L-1}
\end{aligned}$$

□

The bounds for Random Mode are of particular importance: In Random Mode, all send times φ are free variables, considered random. For worst case considerations they take on worst-case parametrizations. Because Random Mode is the most general case and includes all other modes, worst case wait latency of all other modes must also be less than or equal to that of Random Mode:

Proposition 6.2.3.

$$\hat{\Delta} \leq \hat{\Delta}_{\text{Random}},\ \sup\left(\overline{\Delta}\right) \leq \sup\left(\overline{\Delta}_{\text{Random}}\right)$$

Theorem 6.2.4. *Expected mean latency in Random Mode is given by*

$$E\left[\Delta_{\text{Random}}\right] = \frac{1}{2}\sup\left(\overline{\Delta}_{\text{Random}}\right) = \left(2^{L-1}L + 2^L - 1\right)\frac{T}{2\left(2^L-1\right)}$$

Proof. With the receive and send times being uniformly distributed random variables, the expected sojourn time for each relay is $T/2$ which is half of the maximum sojourn time. □

Theorem 6.2.5. *The distribution of traversal time in Random Mode is given by*

$$\mathsf{M}_{\text{Random}} = \sum_{h=1}^{L}\frac{\binom{L}{h}}{2^L-1}\Omega^{*(h+1)} * \chi^{*h} \tag{6.2.1}$$

Proof. For the individual data paths leading to one destination node, traversal time is comprised of different combinations of wait and network delays. These paths are shown in Table 6.2.1 for a network of eight nodes. Ω and χ are random variables denoting wait delay, uniformly distributed between 0 and T, and single hop delay according to Subsection 6.1.1 respectively. The frequencies of occurrences of Ω and χ are defined by the binomial distribution. □

Table 6.2.1: Path traversal times for H_3 in Random Mode

1.	Ω	+	χ									+	Ω
2.				Ω	+	χ						+	Ω
3.	Ω	+	χ	+	Ω	+	χ					+	Ω
4.								Ω	+	χ	+	Ω	
5.	Ω	+	χ				+	Ω	+	χ	+	Ω	
6.				Ω	+	χ	+	Ω	+	χ	+	Ω	
7.	Ω	+	χ	+	Ω	+	χ	+	Ω	+	χ	+	Ω

In Random Mode the overall delay $\breve{\Delta}$ is a function of the offsets of the sampling data procurement phases $\theta_{n,-1}$, the offsets of the send phases of all nodes on all levels $\varphi_{n,\ell}$ and the offsets of the data output phases $\varphi_{n,L}$:

$$\begin{aligned} \breve{\Delta}(\theta_{0,-1}, &\cdots, \theta_{N-1,-1}, \\ \varphi_{0,0}, &\cdots, \varphi_{N-1,0}, \\ &\vdots \\ \varphi_{0,L-1}, &\cdots, \varphi_{N-1,L-1}, \\ \varphi_{0,L}, &\cdots, \varphi_{N-1,L}) \end{aligned} \tag{6.2.2}$$

Assuming sampling data procurement phases $\theta_{n,-1}$ and data output phases $\varphi_{n,L}$ as given, the minimization of overall wait latency is a problem of $2^L \cdot L$ variables. Below we will use Random Mode as a benchmark: Each of the methods intended to reduce wait latency should yield results smaller than the expectation for Random Mode. We start out improving upon Random Mode by attending to obviously avoidable delays.

6.2.2 Sync Mode

Sending data at completely arbitrary phases, regardless of the phases of reception, is not a natural choice. Normally, data is sent as a reaction to some kind of event. An obvious event choice is the reception of data. As an example, consider an audio application. When recording audio, the audio system either calls some audio handler whenever a new buffer of recorded audio data is available or some kind of `read()` or `select()` call on the audio stream file-handle returns. In either case, the control flow proceeds at certain periodic times.

In this timing mode, we use said record or *source event* to trigger the sending of all outgoing packets. Furthermore, we assume that the output or drain phase, i. e. the time at which the aggregated data can be used, e. g. played by a sound-card, is equal to the source phase. This assumption is not unsubstantiated; for instance, many sound-cards synchronize input and output this way. Thus, within each node, all transmissions are triggered at the same time:

$$\varphi_{n,\ell} = \theta_{n,-1}\,,\ \textit{with}\ 0 \le \ell \le L$$

Figure 6.2.2 shows periodic hypercube gossiping in Sync Mode. Note how all of every node's transmissions take place at the same time, subsequent to reception of data from Ł$_{-1}$ (denoted by black tick marks) at an arbitrary phase.

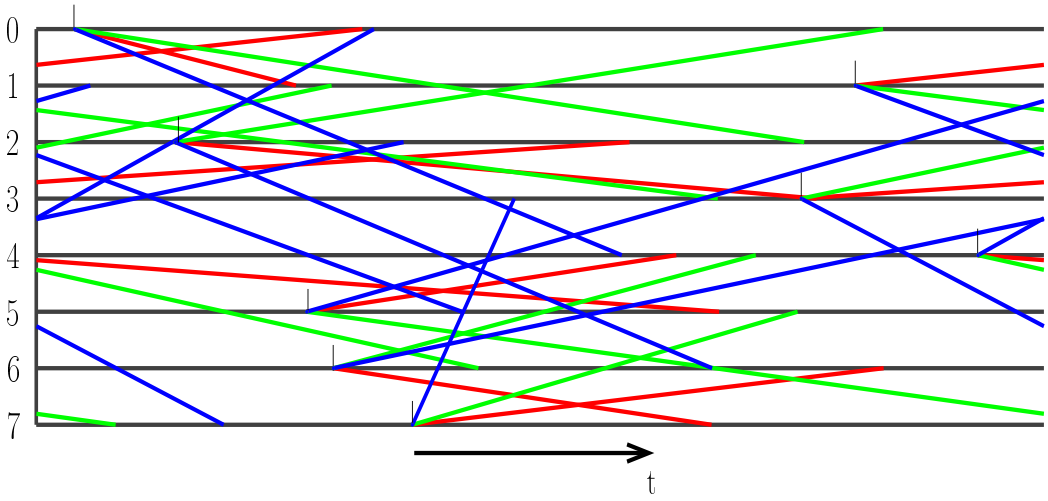


Figure 6.2.2: TSG for H_3 in Sync Mode

Table 6.2.2: Path traversal times for H_3 in Sync Mode

1.	χ									+	Ω
2.					χ					+	Ω
3.	χ	+	Ω	+	χ					+	Ω
4.									χ	+	Ω
5.	χ					+	Ω	+	χ	+	Ω
6.					χ	+	Ω	+	χ	+	Ω
7.	χ	+	Ω	+	χ	+	Ω	+	χ	+	Ω

Consequently, Sync Mode prevents, on each node n, the aging of original sensor data $I(n)$ prior to its transmission:

$$\delta_{n,\ell,-1} = 0$$

Theorem 6.2.6.

$$\hat{\Delta}_{\text{Sync}} = L \cdot T$$

Proof. Synchronizing send phases to procurement phases reduces maximum wait latency by one period because in Random Mode the unsynchronized data source (input) Ł$_{-1}$ contributed one period of wait delay that is now eliminated. □

Corollary 6.2.7.

$$\sup\left(\overline{\Delta}_{\text{Sync}}\right) = \left(2^{L-1}L - 1\right)\frac{T}{2^L - 1}$$

Corollary 6.2.8.

$$E\left[\Delta_{\text{Sync}}\right] = \frac{1}{2}\sup\left(\overline{\Delta}_{\text{Sync}}\right) = \left(2^{L-1}L - 1\right)\frac{T}{2\left(2^L - 1\right)} \tag{6.2.3}$$

All paths, shown in Table 6.2.2, are relieved of their foremost wait delay and traversal time becomes

$$\mathsf{M}_{\text{Sync}} = \sum_{h=1}^{L} \frac{\binom{L}{h}}{2^L - 1} \Omega^{*h} * \chi^{*h} \tag{6.2.4}$$

Note that Sync Mode is–albeit not the best–a completely determined solution, i. e. all send times are specified.

6.2.3 Spliced Mode

Sending all of a node's outgoing packets at the same time is impractical. In a peer-to-peer context, the packets will likely queue up on the Internet uplink and thus increase network latency. Instead of synchronizing the network operations to the source events, Spliced Mode synchronizes all input and output operations to the network. Technically this can be achieved by either

(a) using a separate phase-shifted sensor and sink data stream for each of the node's neighbors if supported by the hardware/device drivers; or at least approximated by

(b) using a sampling period T_{slot} that is a fraction $1/k$ of T and assembling the resulting small packets of sampling data into bigger chunks that will be sent across the network.

If we disregard sub-sample offsets, (a) can be considered a special case of (b) with $k = f_{\text{samp}}$.

Each node has no single sample procurement time $\theta_{n,-1}$ but one $\theta_{n,-1,\ell}$ for each level ℓ, with their phases being apart by integer multiples of $T_{\text{slot}} = T/k$:

$$\theta_{n,-1,v} - \theta_{n,-1,u} = i \cdot T_{\text{slot}}, \quad 0 \le u, v < L,\ i \in \mathbb{Z}.$$

Theorem 6.2.9.

$$\hat{\Delta}_{\text{Spliced}} = (L - 1 + \frac{1}{k})T$$

Proof. Whenever a source node needs to send its recorded samples to any neighbor, it always uses the last packet length of freshly recorded samples. Thereby input related wait delay is eliminated completely:

$$\begin{aligned} \varphi_{n,\ell} &= \theta_{n,-1,\ell}\,,\ 0 \le \ell < L \\ \delta_{n,\ell,-1} &= 0 \end{aligned}$$

Likewise, if the data needs to be output at the destination nodes, the node does not wait until the packets from all levels have arrived but instead uses separate output streams for the different levels, thus permitting to output all received data with little or no additional latency:

$$\delta_{n,L,\ell} \le \frac{1}{k}T$$

Consequently, maximum wait latency is reduced by $(2 - 1/k)$ periods over Random Mode. □

Corollary 6.2.10.

$$\sup\left(\overline{\Delta}_{\text{Spliced}}\right) = \left(\left(\frac{L}{2} - 1 + \frac{1}{k}\right)2^L - 1\right)\frac{T}{2^L - 1}$$

Corollary 6.2.11.

$$E\left[\Delta_{\text{Spliced}}\right] = \frac{1}{2}\sup\left(\overline{\Delta}_{\text{Spliced}}\right) = \left(\left(\frac{L}{2} - 1 + \frac{1}{k}\right)2^L - 1\right)\frac{T}{2\left(2^L - 1\right)}$$

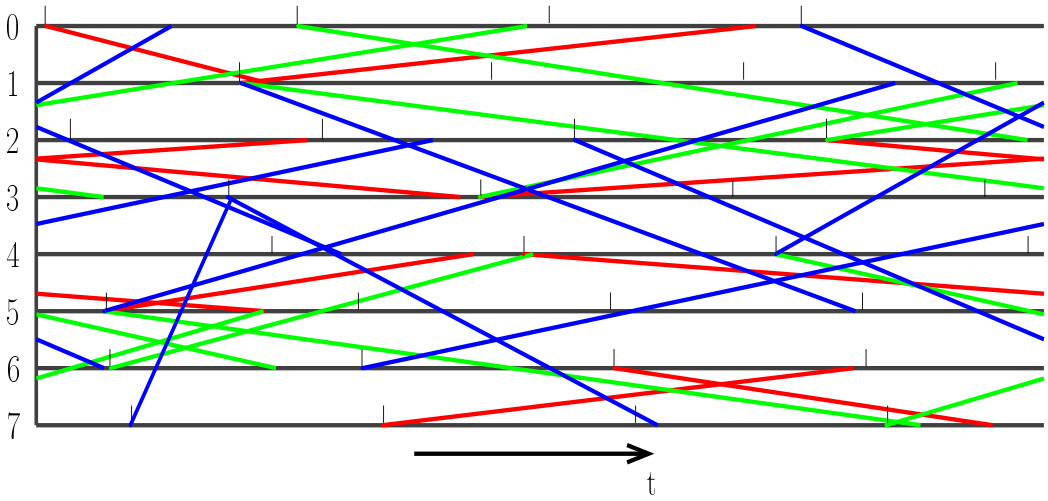


Figure 6.2.3: TSG for H_3 in Spliced Mode, $k = 4$

Table 6.2.3: Path traversal times for H_3 in Spliced Mode

1.	χ									+	ω
2.					χ					+	ω
3.	χ	+	Ω	+	χ					+	ω
4.									χ	+	ω
5.	χ					+	Ω	+	χ	+	ω
6.					χ	+	Ω	+	χ	+	ω
7.	χ	+	Ω	+	χ	+	Ω	+	χ	+	ω

Figure 6.2.3 shows the communication scheme in Spliced Mode with $k = 4$. Note how for each node the send events occur only at four distinct, equidistant phases, denoted by ticks. In contrast to Source/Drain Sync Mode, transmissions are spread out more evenly and the chance of congesting the uplink is decreased.

Comparing Spliced Mode to Sync Mode, it is obvious that Spliced Mode is already a better solution with regard to minimizing overall wait latency. Also, in contrast to Sync Mode, which is a completely determined solution, Spliced Mode still has $2^L(L-1)$ free variables and thus leaves room for further optimizations. Each variable, however, is constrained to k distinct values, rendering the problem discrete. When T_{slot} approaches 0, the send times come completely free like in Random Mode but without the associated delay.

The output related wait delay is divided by k. In Table 6.2.3 this is expressed by ω, denoting a uniform distribution between 0 and T/k. Thus, traversal time becomes

$$\mathsf{M}_{\text{Spliced}} = \sum_{h=1}^{L} \frac{\binom{L}{h}}{2^L - 1} \Omega^{*(h-1)} * \chi^{*h} * \omega \tag{6.2.5}$$

6.2.4 Chained Mode

Remember that each node has exactly one peer at (maximum) hop distance L, as shown in Table 3.3.2 on page 29. Chained Mode refines Spliced Mode by optimizing intermediate send times along longest paths. To reduce the latency for these longest paths as much as possible, Ł$_{\ell+1}$ packets shall be sent promptly after receiving Ł$_\ell$ packets. Like Spliced Mode, Chained Mode synchronizes all node-local input and output operations to the network. The sequence of transmissions is called a chain. Each transmission within a chain is called a chain link. In Figure 6.2.7 on page 99 four such chains are highlighted. A chain connects each node with its longest path peer. All other peers are reached by "skipping" chain links, thereby *switching chains*.

$$\varphi_{n,\ell+1} = \theta_{n,\ell} + \epsilon, \;\; \epsilon \le \frac{T}{k} \quad \Leftrightarrow \quad \delta_{n,\ell+1,\ell} \le \frac{T}{k}$$

Figure 6.2.4 shows the timing in Chained Mode. Note how Ł$_{\ell+1}$ send events happen

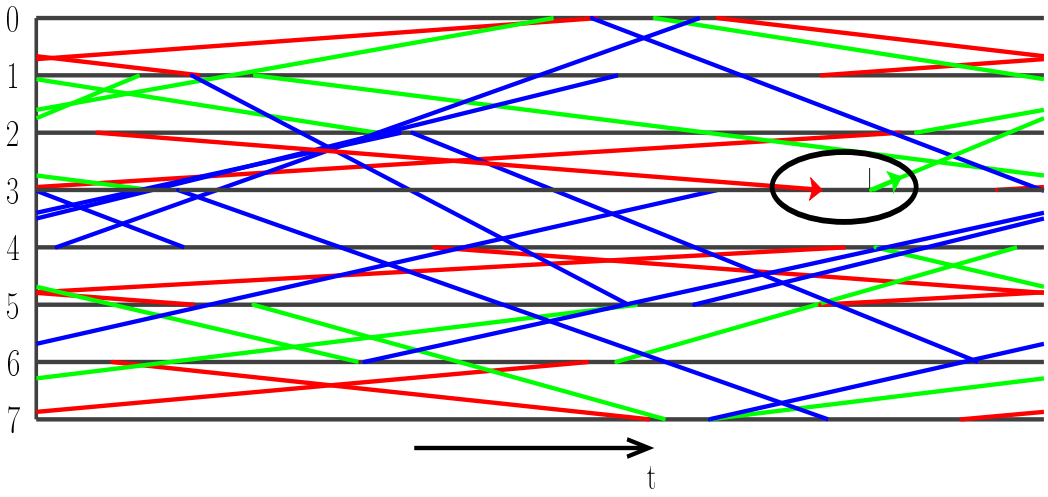


Figure 6.2.4: TSG for H_3 in Chained Mode, $k = 16$

promptly after Ł$_\ell$ receive events, e. g. green departures after red arrivals.

The timing of Chained Mode is thus constituted as follows:

Algorithm 6.1 Timed communication scheme (Chained Mode)

for each $Ł_\ell$ *with* ℓ *in* $[0..L[$ *do loop*

if $\ell > 0$ *then*

wait for $I(Ł_{\ell-1})$ *from neighbor on* $Ł_{\ell-1}$
and mix it into $buffers_{\ell\ldots L-1}$

else

wait until $\varphi_{n,0}$ *(unspecified)*

mix $I(n)$ *from sensor into* $buffer_\ell$

send contents of $buffer_\ell$ *to neighbor on* $Ł_\ell$

Theorem 6.2.12.

$$\hat{\Delta}_{\text{Chained}} = \left(\frac{L}{k} + \left\lfloor \frac{L-1}{2} \right\rfloor \left(1 - \frac{2}{k}\right)\right) T \tag{6.2.6}$$

Proof. Between two chain links of the same chain the maximum wait delay is T_{slot}. The chain is switched by skipping one link. This yields up to one period T of delay instead of two inter-chain-link-delays and can happen every other level. □

Lemma 6.2.13.

$$\sup\left(\overline{\Delta}_{\text{Chained}}\right) = \left(\left(2^{L-1}L - 1\right)\frac{1}{k} + g(L)\left(1 - \frac{1}{k}\right)\right)\frac{T}{2^L - 1}$$

$$\text{with} \quad g(n,\, c=0,\, s=\bot,\, p=0) \to$$

$$\begin{cases} c & when\, n < 1 \\ g(n-1,\, c,\, s,\, 0) + & \\ g(n-1, c + (s \wedge (p=0)\,?\,1 : 0),\, \top,\, 1) & otherwise \end{cases} \tag{6.2.7}$$

Proof. Intra-chain-link-delays are bounded by T_{slot} and occur everywhere except where links are skipped and the delay is one period instead. The recursive function g counts the skip combinations among the node's sources by looking for link-skip-link patterns. $g(n)$ yields the occurrences of 1 followed by any number of 0's followed by 1 in all binary strings of length n. □

In Figure 6.2.5, the function principle of g is illustrated. The recursive function $g(n, c, s, p)$ returns the sum of c and the number of skip combinations in all binary strings of length n that are prefixed by $xxxxp$, where $xxxx$ denotes a string of any number of bits and p is one bit. $s = \top$ means that $xxxx$ contains at least one 1. Figure 6.2.6 shows a plot of $g(x)$.

Theorem 6.2.14.

$$\begin{aligned} E\left[\overline{\Delta}_{\text{Chained}}\right] &= \frac{1}{2}\sup\left(\overline{\Delta}_{\text{Chained}}\right) \\ &= \left(\left(2^{L-1}L - 1\right)\frac{1}{k} + \mathrm{g}(\mathrm{L})\left(1 - \frac{1}{\mathrm{k}}\right)\right)\frac{T}{2\left(2^L - 1\right)} \end{aligned} \tag{6.2.8}$$

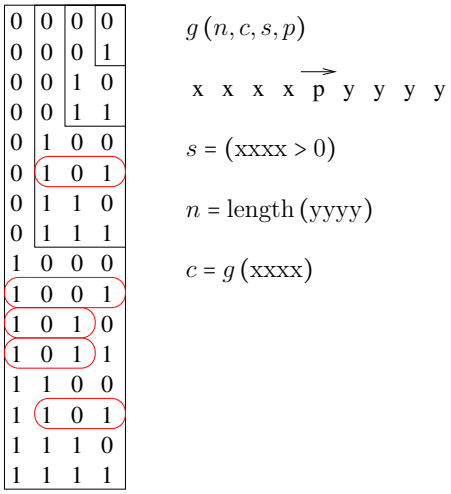


Figure 6.2.5: Function principle of g

Theorem 6.2.15.

$$\mathsf{M}_{\text{Chained}} = \sum_{p=1}^{2^L-1} \frac{1}{2^L-1} \Omega^{*\mathrm{G}(p)} * \omega^{*(\mathrm{H}(p)-\mathrm{G}(p))} * \chi^{*\mathrm{H}(p)}$$

with

$$\mathrm{G}(p, c=0) \to \begin{cases} c & p = 0 \\ \mathrm{G}(\lfloor p/4 \rfloor, c+1) & \lfloor p/4 \rfloor > 0 \wedge (p \bmod 4 = 1) \\ \mathrm{G}(\lfloor p/2 \rfloor, c) & \textit{otherwise} \end{cases}$$

and $\mathrm{H}(p)$ *denoting the Hamming weight of* p.

Proof. For illustration refer to Table 6.2.4, showing the paths for Chained Mode. Intra-

Table 6.2.4: Path traversal times for H_3 in Chained Mode

1.	χ									+	ω
2.					χ					+	ω
3.	χ	+	ω	+	χ					+	ω
4.									χ	+	ω
5.	χ					+	Ω	+	χ	+	ω
6.					χ	+	ω	+	χ	+	ω
7.	χ	+	ω	+	χ	+	ω	+	χ	+	ω

chain-link-delays are ω and occur everywhere except where (as for path 5) links are skipped and the delay is Ω instead. Only when $\Omega > 2\omega + \chi$, path 5 with maximum wait latency (6.2.6) will be the worst-case path instead of (longest) path 7, whose maximum wait latency is

$$\sup_{\varphi,\chi,i} \left(\Delta_{i,i \operatorname{xor}(N-1)} \right) = L \cdot \frac{1}{k} T \tag{6.2.9}$$

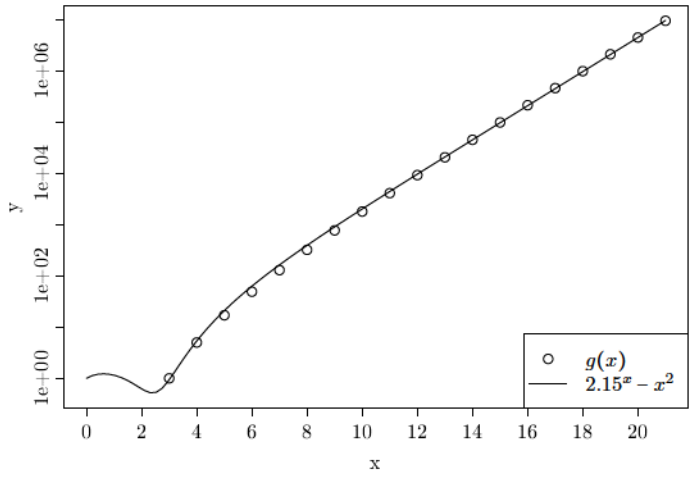


Figure 6.2.6: $g(x)$ and approximation

Hence, the traversal time distribution is characterized not only by the number of transmissions per path but also by their arrangement. Function $\mathsf{G}(p)$ counts the number of gaps in p (i. e. occurrences of the pattern /10+1/ in its binary representation). □

The remaining wait latency ultimately depends on the Ł$_0$ phase offset between Ł$_1$···Ł$_{L-1}$ peers, that is, the problem is reduced to finding a minimum for

$$\breve{\Delta}(\varphi_{0,0}, \cdots, \varphi_{N,0}),$$

which is a minimization problem of just N variables, now.

6.2.5 Crossing Mode

Above timing modes targeting the reduction of latency are limited to node local optimization. By synchronizing the nodes to each other, wait latency may be reduced even further. This requires a terse formulation of the minimization problem. Below, we will investigate the effect of inter-node synchronization on wait latency and examine how the problem of determining optimal transmission times can be reduced to a mixed integer linear programming (MILP) problem.

In Spliced Mode and Chained Mode above, it is assumed that recorded data becomes available at time slots staggered by $1/kT$. Below, we assume for the sake of simplicity and in accordance with Subsection 6.2.3, Case (a), that for any instant t, data recorded during time interval $[t-T, t[$ is readily available, i. e. $k \to \infty$. Thus, between two chain links of the same chain, wait delay is completely eliminated:

$$\varphi_{n,\ell+1} = \theta_{n,\ell},\ \ell \in [0..L-1[\quad \Leftrightarrow \quad \delta_{n,\ell+1,\ell} = 0$$

Decoupling

Spliced Mode removed wait delay after Ł$_{-1}$ and before Ł$_L$. In Chained Mode the remaining wait latency was reduced to chain switching delays, which can only occur on inner levels.

Consequently, wait latency now depends on the Ł$_0$ phase offset between Ł$_1$⋯Ł$_{L-1}$ neighbors. Altering send phase offsets between nodes that are not neighbors on these levels has no effect on wait latency. Through this "decoupling" on Ł$_0$ and Ł$_{L-1}$, the problem of reducing overall wait latency collapses into four independent pieces, henceforth referred to as *partitions*, with $N^* := N/4$ nodes each. We call

$$\alpha_p := \lfloor \frac{p}{2} \rfloor \frac{N}{2} + (p \bmod 2)$$

the *reference node* of the partition p. Wait latency associated with partition p is denoted by Δ_p^*, which is a function of only N^* variables:

$$\breve{\Delta} = \sum_{p=0}^{3} \Delta_p^*(\varphi_{\alpha,0}, \varphi_{\alpha+2,0}, \cdots, \varphi_{\alpha+\frac{N}{2}-2,0})$$

Figure 6.2.7 shows a network with $N = 16$ nodes, i. e. $L = 4$ levels. The first partition is

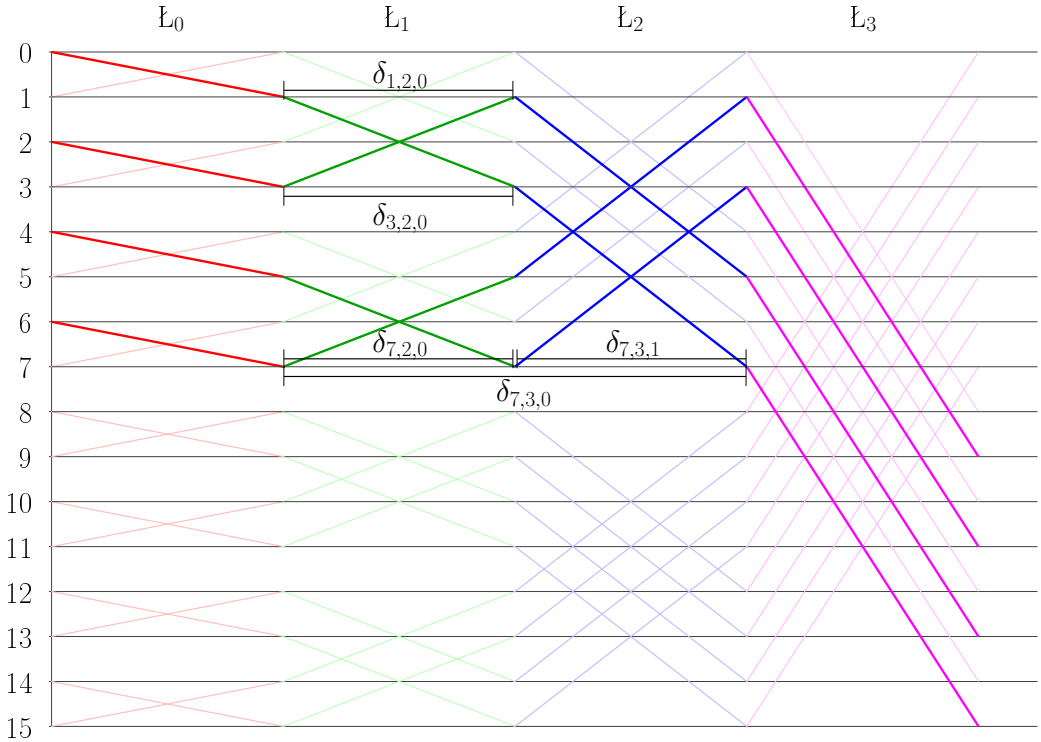


Figure 6.2.7: First partition of 4-level network in Chained Mode.

emphasized. Its latency depends on the relative phase offsets between nodes 0, 2, 4 and 6.

Complementary Wait Delays

Only phase offsets between nodes belonging to the same partition have an effect on wait latency. Now, within one partition, one would expect that shifting one send phase $\varphi_{n,0}$ by an amount z would result in a change in overall wait latency that is proportional to z. Instead, certain ranges or intervals of z result in distinct overall wait latency situations.

Lemma 6.2.16. *Let*

$$z = \varphi_{i,0} \underset{T}{\ominus} \varphi_{j,0}$$

be the phase offset between two nodes' Ł$_0$ send events. Let $\delta_a(z)$ denote some component wait delay that is a non-constant function of z. For each wait delay δ_a there exists a complementary wait delay *δ_b and two intervals $I, I' \subseteq [0, T[$, $I' = \overline{I}$ such that*

$$\begin{aligned} \delta_a(z+y) + \delta_b(z+y) &= \delta_a(z) + \delta_b(z) &< T, \quad y \in I \\ \| \left(\delta_a(z+y') + \delta_b(z+y')\right) - \left(\delta_a(z) + \delta_b(z)\right) \| &= T, \quad y' \in I' \end{aligned}$$

Proof. For illustration refer to Figure 6.2.7, where two wait delays $\delta_{1,2,0}$ and $\delta_{3,2,0}$ are pointed out. Delaying $\varphi_{2,0}$ relative to $\varphi_{0,0}$, i. e. shifting the chain starting at node 2 to the right, will decrease $\delta_{3,2,0}$ but at the same time increase $\delta_{1,2,0}$ by the same amount. In fact, δ_a and δ_b are with regard to each other mirrored and shifted sawtooth functions with both period and amplitude T

$$\begin{aligned} \delta_a(z) &= z \underset{T}{\ominus} a, \\ \delta_b(z) &= -z \underset{T}{\oplus} b \end{aligned}$$

Their sum is a periodic rectangular function as shown in Figure 6.2.8.

$$\delta_a(z) + \delta_b(z) = \begin{cases} b \underset{T}{\ominus} a & \text{for } z \in [a, b[_T \\ \left(b \underset{T}{\ominus} a\right) + T & \text{for } z \in \overline{[a, b[_T} \end{cases}$$

□

In other words, for each pair $\{\delta_a(z), \delta_b(z)\}$ there exists an interval I such that the sum $\delta_a(z) + \delta_b(z)$, $z \in I$ is by T less than for $z \in \overline{I}$. This interval I is referred to as the *"low interval"* (of the rectangle function).

Lemma 6.2.17. *Let nodes a and b be neighbors on Ł$_\ell$ and $\chi_{a,b}$, $\chi_{b,a}$ be the network delays from a to b and b to a, respectively. Then $\delta_a := \delta_{a,\ell+1,\ell-1}$, $\delta_b := \delta_{b,\ell+1,\ell-1}$ are complementary wait delays, and the sum $\delta_a + \delta_b$ is minimal iff $\delta_a, \delta_b \leq \chi_{a,b} \underset{T}{\oplus} \chi_{b,a}$. Also, one implies the other, i. e.*

$$\delta_a \leq \chi_{a,b} \underset{T}{\oplus} \chi_{b,a} \quad \Leftrightarrow \quad \delta_b \leq \chi_{a,b} \underset{T}{\oplus} \chi_{b,a}$$

Proof. To disprove the opposite case, it is, because of Lemma 6.2.16, sufficient to show that $\delta_a + \delta_b > T$:

$$\begin{aligned} & \delta_a > \chi_{a,b} \underset{T}{\oplus} \chi_{b,a} \\ \Rightarrow\ & \delta_a > \delta_a \underset{T}{\ominus} \left(\chi_{a,b} \underset{T}{\oplus} \chi_{b,a}\right) = \delta_a - \left(\chi_{a,b} \underset{T}{\oplus} \chi_{b,a}\right) \end{aligned}$$

Because $\delta_b = \left(\chi_{a,b} \underset{T}{\oplus} \chi_{b,a}\right) \underset{T}{\ominus} \delta_a$ and $x \underset{T}{\ominus} y = T \underset{T}{\ominus} \left(y \underset{T}{\ominus} x\right)$, it follows

$$\begin{aligned} & \Rightarrow \delta_a > \delta_a \underset{T}{\ominus} \left(\chi_{a,b} \underset{T}{\oplus} \chi_{b,a}\right) = \delta_a - \left(\chi_{a,b} \underset{T}{\oplus} \chi_{b,a}\right) = T - \delta_b \\ & \Rightarrow \delta_a + \delta_b > T \\ & \Rightarrow \delta_a + \delta_b \text{ is not minimal.} \end{aligned}$$

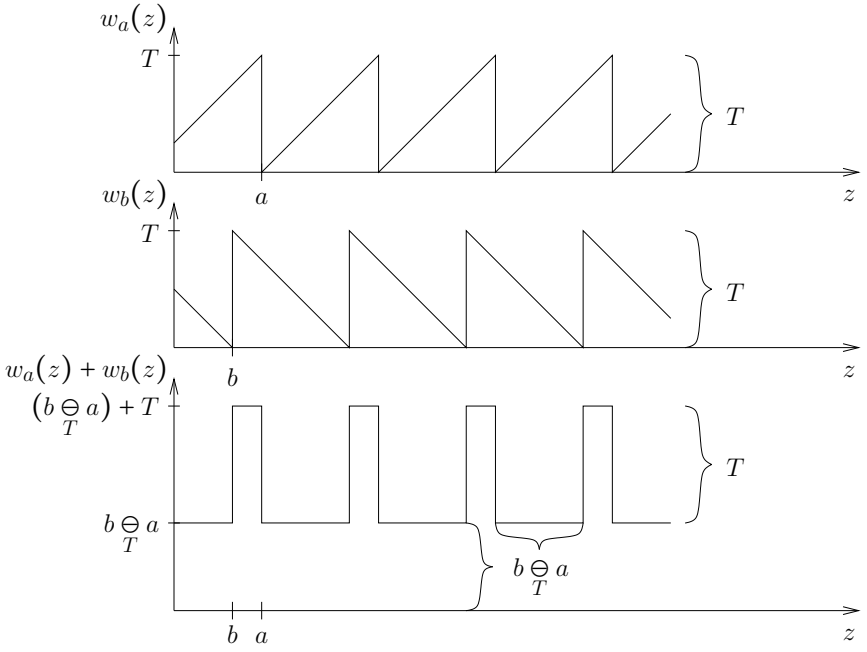


Figure 6.2.8: The sum of two complementary delays

For the reverse, we consider:

$$\begin{aligned}
&\delta_b + \delta_a > T \\
\Leftrightarrow\ &((T - \delta_a) + \chi_{a,b} + \chi_{b,a}) \bmod T + \delta_a > \delta_a + (T - \delta_a) \\
\Leftrightarrow\ &((T - \delta_a) + \chi_{a,b} + \chi_{b,a}) \bmod T > T - \delta_a \\
\Rightarrow\ &x < T - (T - \delta_a) = \delta_a
\end{aligned}$$

The same applies for δ_b. □

Refer to Figure 6.2.9 for an illustration: $\delta_{1,2,0}$ is greater than the sum of $\chi_{1,3}$ and $\chi_{3,1}$; hence, their subtraction does not wrap and the difference, equaling $(\varphi_{3,1} - \theta_{3,1}) \bmod T = T - \delta_{3,2,0}$, is less than the minuend. Therefore, $\delta_{1,2,0} + \delta_{3,2,0} > T$. According to Lemma 6.2.16, the sum cannot be minimal if it is greater than T. For $\delta_{5,2,0}$ and $\delta_{7,2,0}$, the communication is timed in such a way that the Ł$_1$ transmissions overlap, i. e. "cross" each other. The subtraction wraps, $T - \delta_{7,2,0}$ is not less than $\delta_{5,2,0}$, and the $\delta_{5,2,0} + \delta_{7,2,0}$ sum is less than T and thus minimal. Note how shifting $\varphi_{2,0}$ (and thereby $\varphi_{3,1}$ and $\theta_{1,1}$) to the left would make $\chi_{1,3}$ and $\chi_{3,1}$ cross, too.

For each partition, Δ_p^* is minimal if the send phase offsets are chosen in such a way that all sums of complementary delays are minimal, i. e. x lies in the low interval. This condition is described by a system of range constraints as follows. Within a partition, each node is involved in

$$d = \sum_{\ell=1}^{L-2} L - \ell - 1 = \frac{1}{2}(L-1)(L-2)$$

range constraints. Note that wait delay is not limited to consecutive levels. E.g. for node 7 of Figure 6.2.7, constraints comprise wait delays $\delta_{7,2,0}$, $\delta_{7,3,0}$, and $\delta_{7,3,1}$. Particularly,

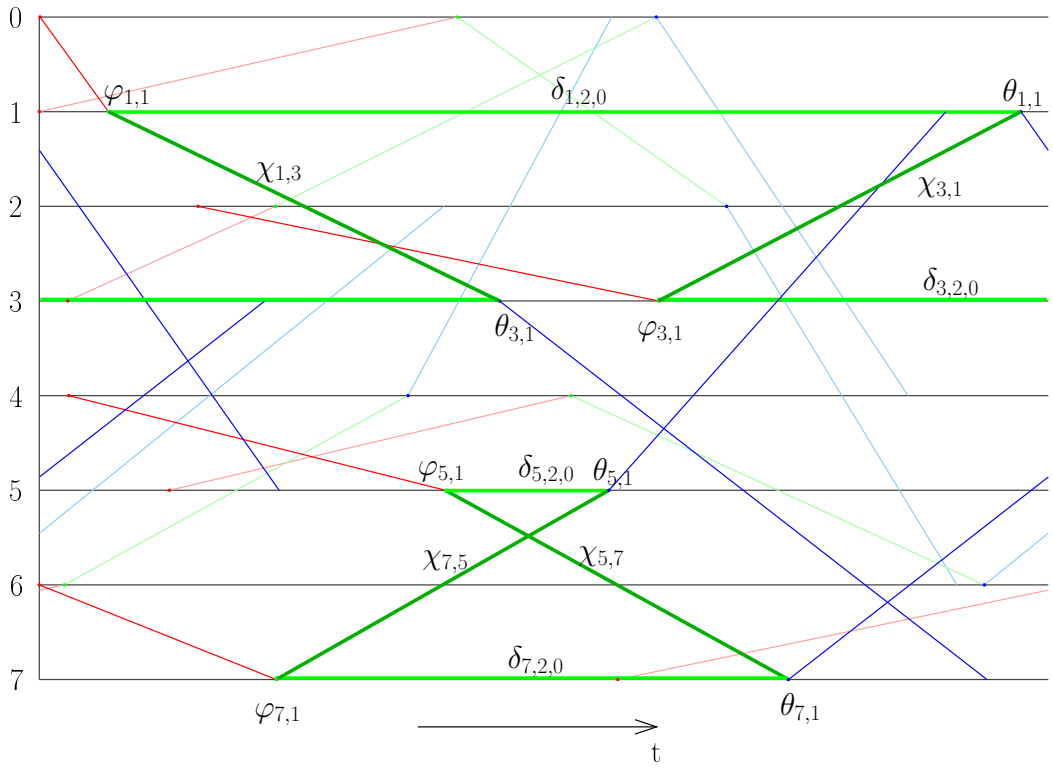


Figure 6.2.9: Network with 8 nodes in Chained Mode. The sum of complementary delays $\delta_{5,2,0}$ and $\delta_{7,2,0}$ is minimal, the one of $\delta_{1,2,0}$ and $\delta_{3,2,0}$ is not.

the interdependencies between one partition's nodes form a d-regular graph with N^* nodes and $|E| = \frac{N^* d}{2}$ edges. Each edge e defines a range constraint for the offset z_e between two nodes:

$$z_e \in [a_e, b_e[_T \tag{6.2.10}$$

Let n_0 and n_1 be two nodes connected by edge e. Then parameters a_e and b_e can be computed by traversing the chains starting at n_0 and n_1, respectively, and summing network delays along the way. The principle algorithm is shown in Algorithm 6.2. The chains will meet at two nodes, called *rendezvous nodes*. At these points, the modulo of the sums' difference equals a_e or b_e, depending on the order of levels of the last chain link that brought each chain there. Note that as each node has only one incoming link per level, the levels cannot be equal. However, Algorithm 6.2 disregards slot wait delays incurred at each intermediate message arrival; Algorithm 6.3 takes these into account, too. For instance, the interval boundaries for $\varphi_{2,0} \ominus_T \varphi_{0,0}$, i. e. the first interval constraint of partition 0 with rendezvous $\varphi_{1,2} \ominus_T \varphi_{1,1}$ and $\varphi_{3,2} \ominus_T \varphi_{3,1}$, are determined as

$$a = \Big(\chi_{0,1} - \big((\theta_{0,-1} + \chi_{0,1} - \theta_{1,-1}) \bmod T_{\text{slot}} \big) \\ - \Big(\chi_{2,3} + \big((\theta_{3,-1} - (\theta_{2,-1} + \chi_{2,3})) \bmod T_{\text{slot}} \big) + \chi_{3,1} \Big) \Big) \bmod T$$

$$b = \Big(\chi_{0,1} + \big((\theta_{1,-1} - (\theta_{0,-1} + \chi_{0,1})) \bmod T_{\text{slot}} \big) + \chi_{1,3} \\ + \Big(\Big(\theta_{3,-1} - \Big(\theta_{0,-1} + \chi_{0,1} + \big((\theta_{1,-1} - (\theta_{0,-1} + \chi_{0,1})) \bmod T_{\text{slot}} \big) + \chi_{1,3} \Big) \Big) \bmod T_{\text{slot}} \Big) \\ - \chi_{2,3} \Big) \bmod T$$

Each range constraint (6.2.10) defines a subspace, referred to as *slice,* in modulo space, delimited by two parallel (supporting) hyperplanes. The cut-set of slices corresponds to a system of inequations, each containing modulo operators. In order to solve this system in non-modulo space, the modulo operators need to be eliminated using case differentiations as shown in Subsection 3.4.1. As variables of the system we choose not send phases φ directly but rather *relative send phase offsets* with regard to the reference node. This way, we manage with one variable less and avoid redundant solutions and the risk of degeneracy and cycling of the MILP solver caused by such superfluous variables. Let x_i, called *primary offset,* denote the offset to the send phase of the reference node α of the partition:

$$x_i := \varphi_{\alpha+2(i+1),0} \underset{T}{\ominus} \varphi_{\alpha,0},\ i \in [0..N^* - 1[\tag{6.2.11}$$

Algorithm 6.2 Computation of parameters a and b (assuming $T_{\text{slot}} = 0$)

```
sum_0, sum_1 = 0
for (ℓ_0 in 0...L) // traverse chain_0 starting at n_0
    nb_0 = nb(n_0, ℓ_0)
    sum_0 += χ_{n_0,nb_0}
    n_0 = nb_0
    for (ℓ_1 in 0...L) // traverse chain_1 starting at n_1
        nb_1 = nb(n_1, ℓ_1)
        sum_1 += χ_{n_1,nb_1}
        n_1 = nb_1
        if (n_1 == n_0) // rendezvous
            if (ℓ_1 > ℓ_0)
                a = (sum_0 − sum_1) mod T
            else
                b = (sum_0 − sum_1) mod T
```

Algorithm 6.3 Computation of parameters a and b regarding slot wait delays

$start_0 = true$
for (ℓ_0 in $0...L$) // *traverse* $chain_0$ *starting at* n_0
  $nb_0 = \mathrm{nb}(n_0, \ell_0)$
  if ($start_0$)
    $sum_0 = \chi_{n_0,nb_0}$
    $sum'_0 = (\theta_{n_0,-1} + \chi_{n_0,nb_0}) \bmod T_{\mathrm{slot}}$
  else
    $slotwait_0 = (\theta_{n_0,-1} - sum'_0) \bmod T_{\mathrm{slot}}$
    $sum_0 += slotwait_0 + \chi_{n_0,nb_0}$
    $sum'_0 += slotwait_0 + \chi_{n_0,nb_0}$
  $n_0 = nb_0$
  $start_1 = true$
  for (ℓ_1 in $0...L$) // *traverse* $chain_1$ *starting at* n_1
    $nb_1 = \mathrm{nb}(n_1, \ell_1)$
    if ($start_1$)
      $sum_1 = \chi_{n_1,nb_1}$
      $sum'_1 = (\theta_{n_1,-1} + \chi_{n_1,nb_1}) \bmod T_{\mathrm{slot}}$
    else
      $slotwait_1 = (\theta_{n_1,-1} - sum'_1) \bmod T_{\mathrm{slot}}$
      $sum_1 += slotwait_1 + \chi_{n_1,nb_1}$
      $sum'_1 += slotwait_1 + \chi_{n_1,nb_1}$
    $n_1 = nb_1$
    if ($n_1 == n_0$) // *rendezvous*
      if ($\ell_1 > \ell_0$)
        $a = (sum_0 - ((sum'_0 - \theta_{n_1,-1}) \bmod T_{\mathrm{slot}}) - sum_1) \bmod T$
      else
        $b = (sum_0 - ((\theta_{n_1,-1} - sum'_0) \bmod T_{\mathrm{slot}}) - sum_1) \bmod T$

All phase offsets z_k can then be expressed in terms of either primary offsets x_i or differences thereof, referred to as *secondary offsets*:

$$z_k \in \{x_i \,|\, i \in [0..N^*[\} \;\cup\; \{x_j - x_i \,|\, i,j \in [0..N^*[\wedge j > i\}$$

For instance, primary offset x_0 of partition 0 is defined as the difference $\varphi_{2,0} \ominus \varphi_{0,0}$.

Primary Offsets

We first consider the case of z being a primary offset to α.

$$z \in \{x_i \,|\, i \in [0..N^*[\}, z, a, b, x \in [0, T[\;\Rightarrow\; z \in [0,\, T[$$
$$\Rightarrow\; b \underset{T}{\ominus} z = \begin{cases} b - z & \text{for } 0 \le z \le b \\ b - z + T & \text{for } b < z < T \end{cases} \qquad (6.2.12)$$

For the solutions to (6.2.10) we discern two cases with regard to b and a, depicted in Figures 6.2.10 and 6.2.11:

Case $b \ge a$ **:** In this trivial case, displayed in Figure 6.2.10, the interval is completely contained in $[0, T[$ and hence the solution in non-modulo space contiguous, too. To satisfy equation (6.2.10), z must lie between a and b:

$$b \ge a \Rightarrow \qquad (b-a) \bmod T \;=\; b - a, \text{ by (6.2.10)}$$

$$0 \le z \le b: \qquad \begin{aligned} b - z &\le b - a \\ z &\ge a \\ S &= \{z \,|\, a \le z \le b\} \end{aligned}$$

$$b < z < T: \qquad \begin{aligned} b - z + T &\le b - a \\ z - T &\ge a \\ S &= \{\} \end{aligned}$$

i. e.

$$\begin{aligned} z &\ge a \\ z &\le b \end{aligned}$$

In order to obtain a system in canonical form, we reverse the first inequality by multiplying with -1:

$$\begin{aligned} -z &\le -a \\ z &\le b \end{aligned} \qquad (6.2.13)$$

Case $b < a$ **:** In this case, shown in Figure 6.2.11, the solution interval of (6.2.13) is "wrapped around" the boundaries of the codomain and therefore falls into two disjunct solution sets. A value z belongs to the solution set iff it is either smaller than b or greater than a:

$$\begin{aligned} -z &\le -a \\ &\vee \\ z &\le b \end{aligned}$$

The disjunction can be transformed into a conjunction by introducing a binary case variable $c \in \{0; 1\}$ with oppositional signed coefficients for the conflicting portions. The absolute

value of the coefficient must be large enough to lever out the respective row. We use $\pm T$. A value of 1 makes the first row of (6.2.14) instantly feasible, a value of 0 the second:

$$\begin{array}{rrcl} -z & -T\cdot c & \leq & -a \\ z & +T\cdot c & \leq & b+T \end{array} \tag{6.2.14}$$

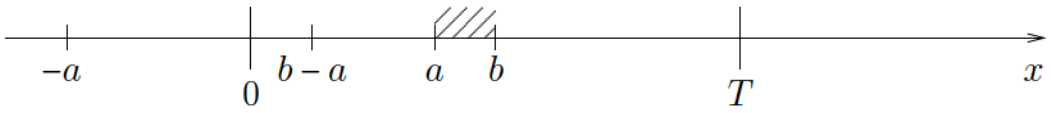


Figure 6.2.10: 1-Variable-Slice in modulo-T parameter space, case $b \geq a$

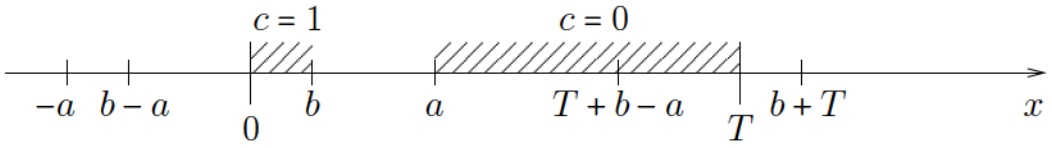


Figure 6.2.11: 1-Variable-Slice in modulo-T parameter space, case $b < a$

Secondary Offsets

The remaining (secondary) phase offsets z_k are expressed in terms of differences of primary offsets:

$$z_k \in \{x_j - x_i \,|\, i,j \in [0..N^*[\wedge j > i\}$$

Note that although x_i and x_j are both positive, the codomain of z now extends into the negative:

$$x_i, x_j \in [0,T[\quad \Rightarrow \quad z_k \in]-T,T[$$

Because of the periodicity of the modulo space, the extension of the codomain yields additional solutions that must be considered. This can be achieved by eliminating the modulo operator as follows: $z \in]-T,T[\Rightarrow$

$$b \underset{T}{\ominus} z = \begin{cases} b-z-T & \text{for} \quad -T < z \leq b-T \\ b-z & \text{for} \quad b-T < z \leq b \\ b-z+T & \text{for} \quad b < z < T \end{cases}$$

Again, there are two cases with regard to b and a but this time with two and three disjunct solution sets, respectively. The solution to (6.2.10) is extended by a shifted copy of the original solution set from Subsection 6.2.5:

$$S' = S \cup \{z - T \,|\, z \in S\}$$

Figure 6.2.12 shows the solutions to (6.2.10) for $z = x_1 - x_0$.

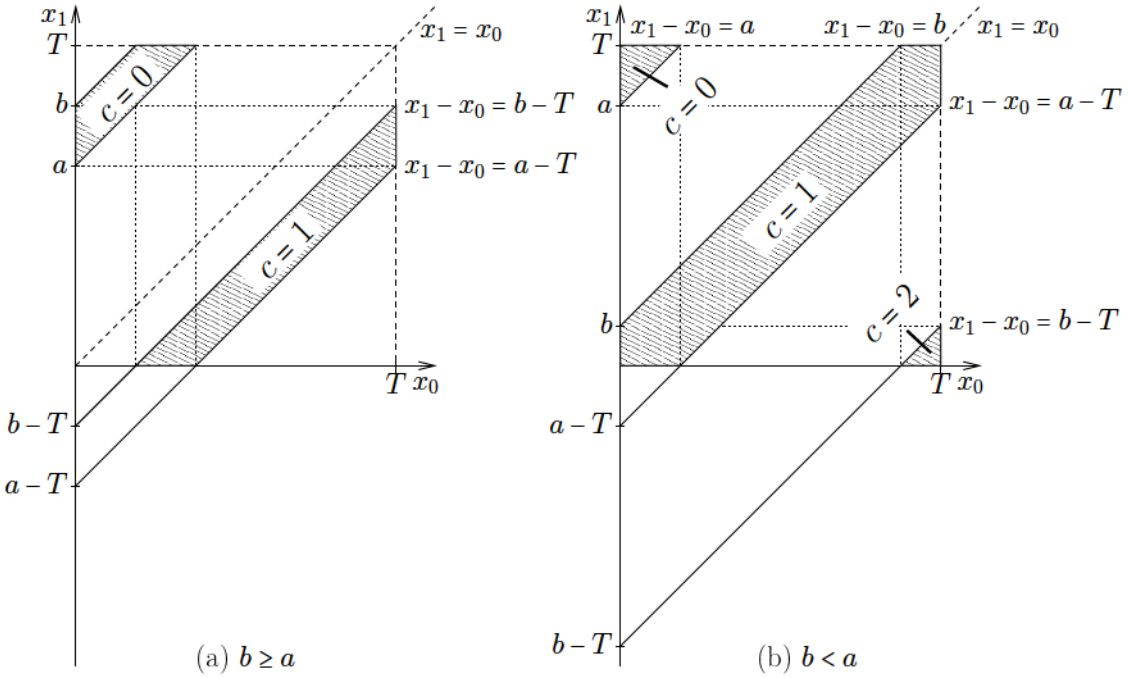


Figure 6.2.12: 2-variable-slice in modulo-T parameter space: Two cases

Case $b \geq a$: As shown in Figure 6.2.12a, the formerly contiguous solution of (6.2.13) is complemented by its image shifted by $-T$, thus yielding two disjunct solution areas

$$\begin{array}{rcl} -z & \leq & -a \\ z & \leq & b \\ & \vee & \\ -z & \leq & T - a \\ z & \leq & b - T \end{array}$$

Again, we use a binary case variable $c \in \{0; 1\}$ to transform the disjunction into a conjunction:

$$\begin{array}{rrcr} -z & -T \cdot c & \leq & -a \\ z & -T \cdot c & \leq & b \\ -z & +T \cdot c & \leq & 2T - a \\ z & +T \cdot c & \leq & b \end{array} \tag{6.2.15}$$

For $c = 1$, row 1 and 3 are equivalent. Row 4 is a tighter constraint than row 2. Removing rows 2 and 3 leaves

$$\begin{array}{rrcr} -z & -T \cdot c & \leq & -a \\ z & +T \cdot c & \leq & b \end{array} \tag{6.2.16}$$

Case $b < a$: When the formerly disjunct slice (6.2.14) in Subsection 6.2.5 is duplicated, the overlapping pieces are rejoined, creating a total of not four but three disjunct solution

areas.

$$\begin{array}{rcl} -z & \leq & -a \\ & \vee & \\ -z & \leq & T-a \\ z & \leq & b \\ & \vee & \\ z & \leq & b-T \end{array}$$

The three cases can be modeled with a single integer variable $c \in \{0,1,2\}$.

$$\begin{array}{rrcl} -z & -T \cdot c & \leq & -a \\ z & +T \cdot c & \leq & b+T \end{array}$$

Using the REEL tool presented in Chapter 5, the conditions required to satisfy each range constraint can be explored. Particularly, the problems view shown in Figure 5.5.2 on page 79 provides this information in numeric as well as symbolic form.

The Shape of the Perfect Solution

Sought is the vector $\mathbf{x}$ of phase offsets between peers that makes all sums of complementary delays minimal. Each system of inequalities defines one or more convex, bounded polytopes in (N^*-1)-dimensional modulo-T space $\{x \in \mathbb{R} \,|\, 0 \leq x < T\}^{L-1}$. Every point that lies within one of the polytopes is an equivalent, perfect solution to the minimization problem. The right sides, however, contain quantile estimates of stochastic network delays, that may also change slightly over time. Optimally, the aspired solution should therefore be located as far from the boundary of the polytope as possible. This can be achieved by determining the center of the largest inscribed sphere of all polytopes. The center can be obtained by shrinking each polytope, i. e. shifting all facets by the same amount along their normals towards the inside of the polytope. The maximum shifting distance before the polytope disappears is then the radius of the maximum sphere fitting inside. Depending on the number of points where the sphere touches the polytope, the resulting body after the shrinking will have at least one dimension less than the original polytope.

To check whether a feasible solution to a linear program exists, Phase-1 of the *Simplex Algorithm* [39] can be used [138]. For enumerating the vertices of the (shrunk) polytope [4] can be used.

Relaxation and Optimization

A solution where all partial sums are minimal may not exist. Hence, we search for solutions violating as few constraints as possible. Moreover, different constraints, when violated, lead to different latency penalties. The number of paths that contain one particular wait delay δ determines the weight of the violation. The weight doubles with every chain link that is taken. Hence, it can be computed from the size of the gap $(r-l)$ with r and l denoting the right and left borders of the gap, respectively.

$$w = 2^{L-(r-l)-2}$$

To find an optimal solution, we define an objective function to be minimized and one binary relaxation variable p_e for each constraint by which each it may be overridden. The weights are used as coefficients to relaxation variables p_e within the objective function.

$$obj = T \sum w_e p_e$$

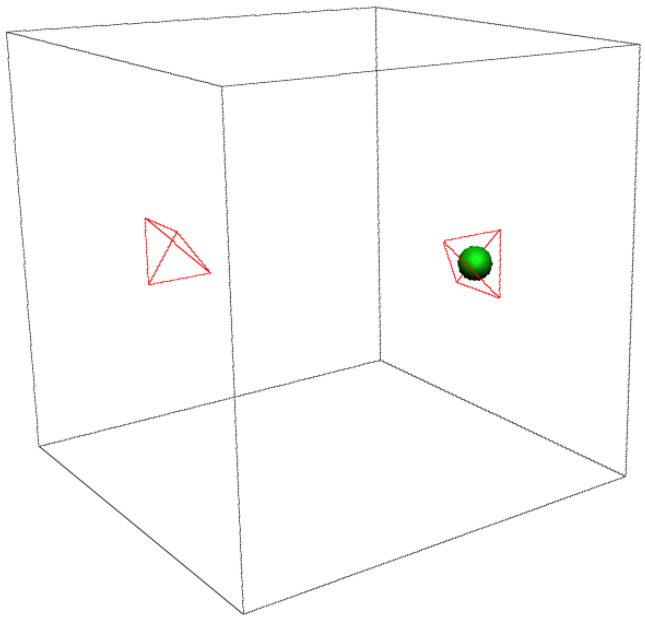


Figure 6.2.13: Largest sphere (green) fitting minimum delay solution polytopes (red) in modulo-T parameter space (gray).

Each range constraint is then represented as follows, depending on whether z is a primary or secondary offset and on the order of a and b:

Case z is primary offset:

Case $b > a$:

$$\begin{array}{rrrcl} -z & & -T\cdot p & \leq & -a \\ z & & -T\cdot p & \leq & b \end{array}$$

Case $b < a$:

$$\begin{array}{rrrcl} -z & -T\cdot c & -T\cdot p & \leq & -a \\ z & +T\cdot c & -T\cdot p & \leq & b+T \\ & c & & \in & \{0,1\} \end{array}$$

Case z is secondary offset:

Case $b > a$:

$$\begin{array}{rrrcl} -z & -T\cdot c & -T\cdot p & \leq & -a \\ z & +T\cdot c & -T\cdot p & \leq & b \\ & c & & \in & \{0,1\} \end{array}$$

Case $b < a$:

$$\begin{array}{rrrcl} -z & -T{\cdot}c & -T\cdot p & \leq & -a \\ z & +T{\cdot}c & -T\cdot p & \leq & b+T \\ & c & & \in & \{0,1,2\} \end{array}$$

6.3 Evaluation

6.3.1 Random, Sync, Spliced, Chained Mode

Overall Wait Latency $\breve{\Delta}$, Upper Bound $\hat{\Delta}$ and Expectation $E[\Delta]$

Having derived analytic bounds and expectations, the gossiping scheme was emulated on the REEL tool, presented in Chapter 5, with 128 nodes for the first four timing modes, initially with a hypothetical constant network delay of 50 ms, then with plausible German network delays. The period T was set to 100 ms. Figure 6.3.1 shows the quantile functions for the three latency components wait latency (black solid, top) and network latency (green striped, bottom). The area of each component represents its overall latency (i. e. $\breve{\Delta}$ for wait latency). The red dotted/dashed horizontal lines show the derived theoretical bounds $\hat{\Delta}$ and expected value $E[\Delta]$, respectively, specific to each method. The solid black horizontal line (partly occluded by the red dashed line) shows the mean of the simulated wait delay values. Constant network delay was chosen over uniform to reveal the hop count distribution presented in Subsection 3.3.2. In Figure 6.3.2, the constant network delay was replaced by delays obtained from the `GlobalNetworkPositioningDelayModel` model of the PeerfactSim.KOM simulator [119], that realistically reproduces network locality. The parametrization was done with measurement data obtained from `speedtest.net` as detailed in Section 4.5. Results show a reduction of mean wait latency by roughly two thirds and maximum wait latency by half.

Traversal Time M

To verify the analytical traversal time estimates, the gossiping scheme was emulated with varying network sizes and values of k for the first four timing modes, using the hop delay distribution shown in Figure 6.1.1. The period T was set to 840 ms to allow for a variety of divisors. Figure 6.3.3 shows the 99^{th} percentile of traversal time for each method as a function of the number of nodes with k set to 8. The analytical estimate closely approximates the simulation results. For Sync and Spliced Mode a constant reduction can be observed with regard to Random Mode. For Chained Mode the reduction is proportional to the number of nodes. In Figure 6.3.4, traversal time is plotted over k. Obviously, Random Mode and Sync Mode are only defined for $k = 1$ and merely serve as a reference. As can be seen with regard to Spliced Mode and Chained Mode, the real benefit of I/O splicing is not the latency reduction in Spliced Mode itself ($\sim 0.5T$) but rather the laying of foundations for Chained Mode and its reduction of approximately $2T$ for 64 nodes.

6.3.2 Crossing Mode

To evaluate the additional reduction of latency by the global optimization approach Crossing Mode, it too was integrated into the REEL tool. To this end, the operational model was extended by routines for the computation of parameters a and b and the construction

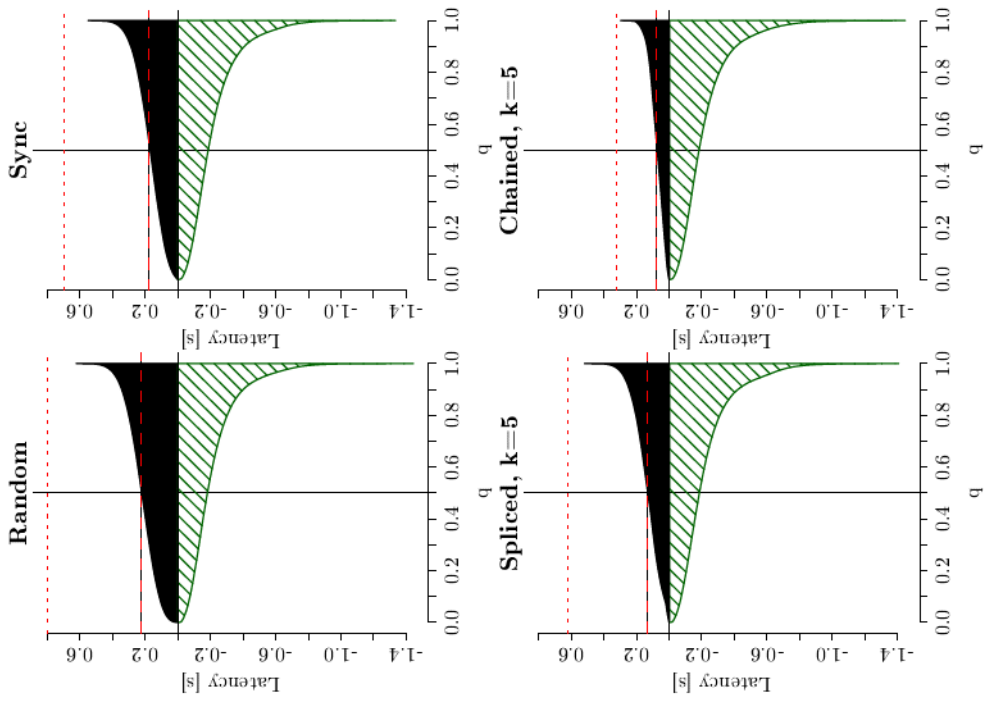


Figure 6.3.2: Quantile functions of wait and network latency for sampled hop delays (domestic connections in Germany), $N = 128$, $T = 100\,\text{ms}$

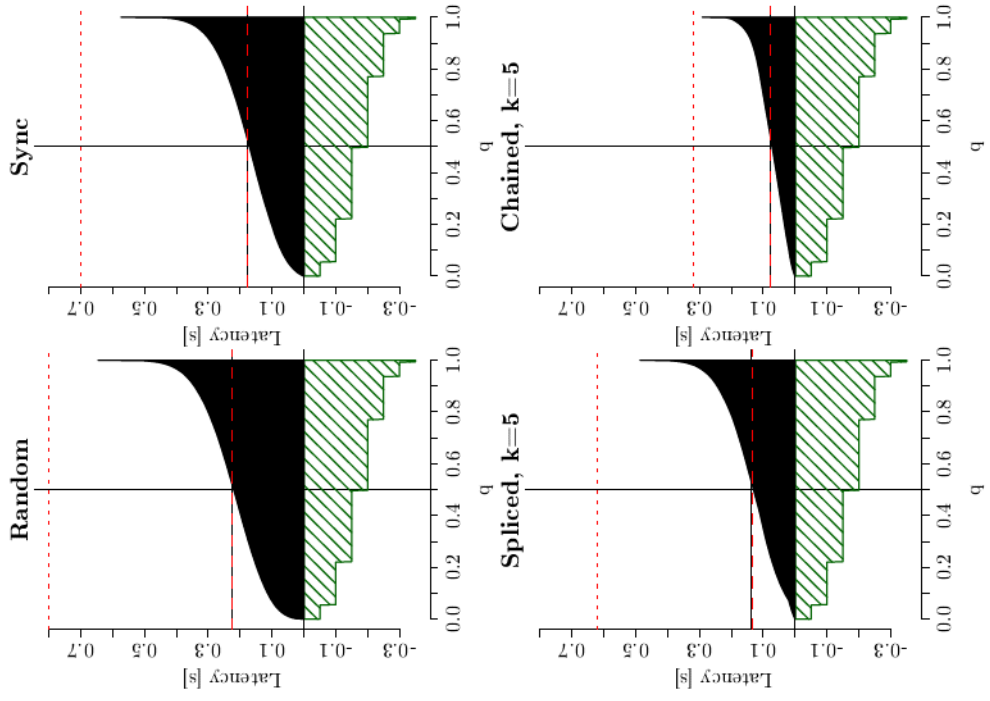


Figure 6.3.1: Quantile functions of wait and network latency for constant hop delay (50 ms), $N = 128$, $T = 100\,\text{ms}$

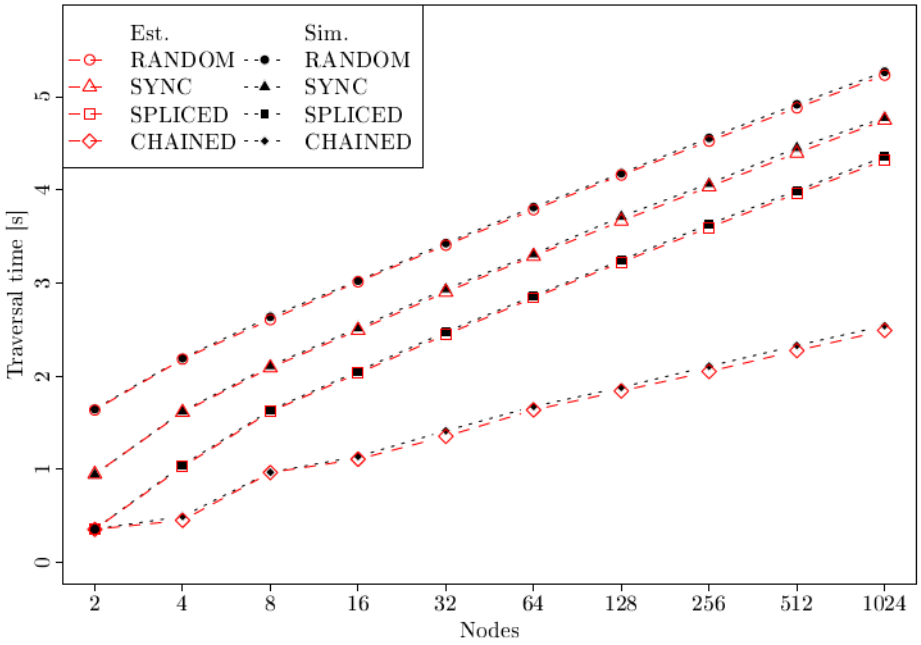


Figure 6.3.3: 99^{th} percentile of traversal-time as a function of network size, $T = 840\,\text{ms}$, $k = 8$

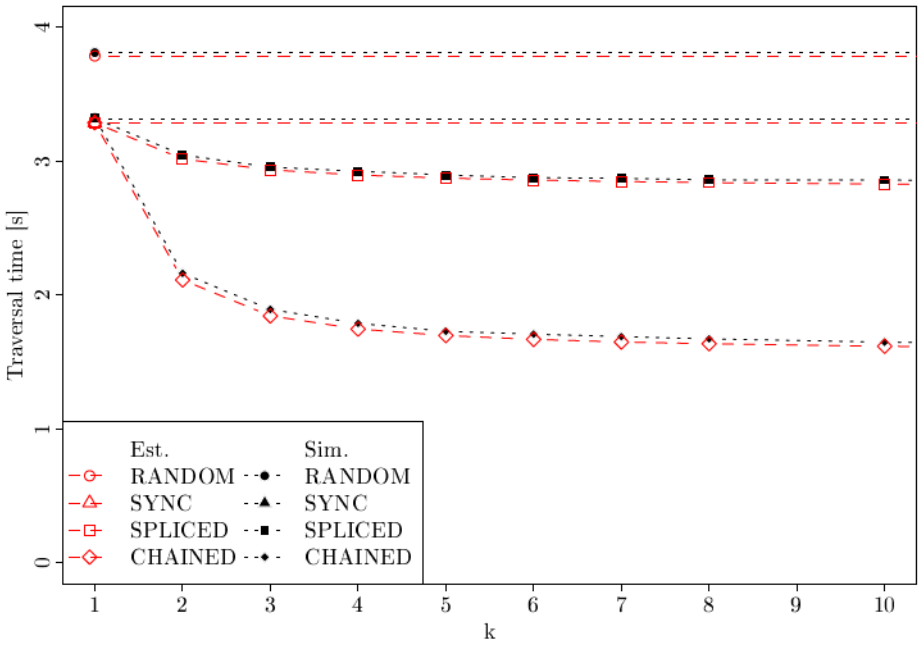


Figure 6.3.4: 99^{th} percentile of traversal-time as a function of chunking parameter k, $T = 840\,\text{ms}$,$N = 64$

of the system of inequalities as laid out in Subsection 6.2.5. For the solution of the system, the open source MILP solver `lp_solve` [11] was linked to the JAVA-based emulator by means of a JNI wrapper. `lp_solve` is based on the revised simplex method [40] and the uses branch-and-bound search for the integer constraints. Unfortunately, attempts to set solution parameters through the JNI wrapper caused the optimizer to crash. Because solution performance heavily depends on these parameters, the integration is only of interest for interactive use with small models of up to 32 nodes. For experiments with larger models, the wrapper was only used to write the model in LP format to disk. The actual solution was performed with the stand-alone executable `lp_solve` and the faster, commercial GUROBI OPTIMIZER [60]. For comparability's sake, $\chi \bmod T$ was assumed to be uniformly distributed over T, thereby obviating the need for further assumptions regarding the actual magnitude of network delay. Network size was varied between 8 and 256 nodes. Delay configurations were generated randomly subject to 20 different seeds. Figure 6.3.5 shows mean wait latency $\overline{\Delta}$ in Crossing Mode in relation to Chained Mode and subject to network size. The cyan dot-dashed line at the top marks the reference, i. e. wait latency for a network of given size in Chained Mode. The black line shows the hypothetical wait latency that will result if the configuration from Chained Mode can be solved perfectly, i. e. if the objective function becomes zero. As we can see, this would on average result in the halving of wait latency. However, a perfect solution that meets all constraints is not always possible (see Subsection 6.2.5). The red line shows the optimal solutions that can be obtained on average. While the solution of the MILP for networks of 32 nodes

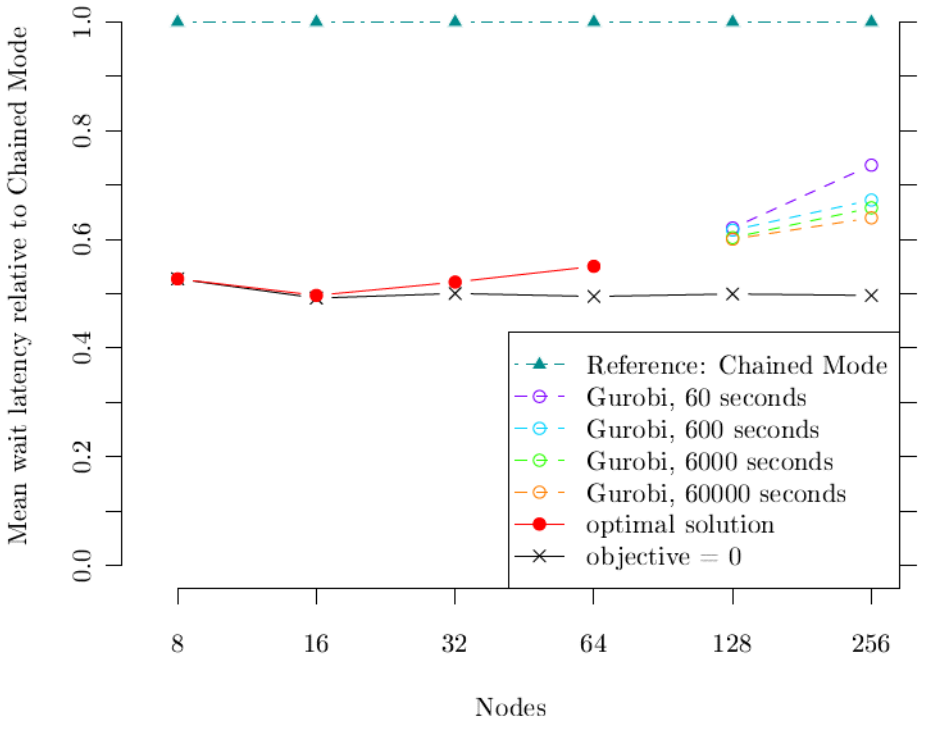


Figure 6.3.5: Reduction of wait latency in Crossing Mode relative to Chained Mode, subject to network size

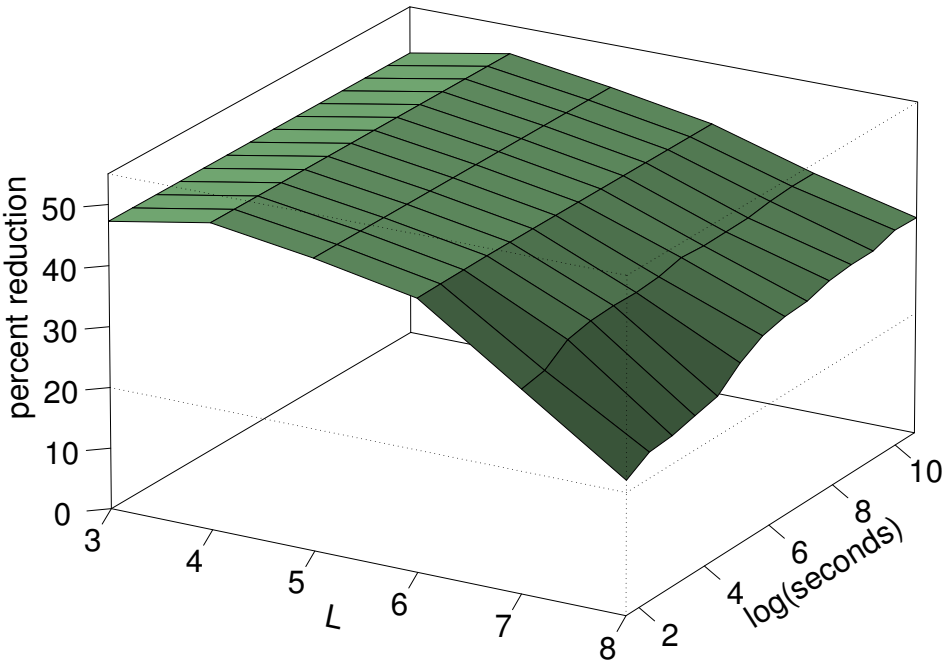


Figure 6.3.6: Reduction of wait latency in Crossing Mode subject to network size and solution time

took only a few milliseconds on a single 2.8 GHz CPU core, the complete solution for a network of 64 nodes took already several seconds, whereas perfect solutions for 128 nodes and more were unobtainable within 60,000 s (16.7 h) solution time. For networks of 128 and 256 nodes, the hollow circles depict solutions that can be obtained in various orders of solution time. The same results are shown in Figure 6.3.6 in more detail; this time the *reduction* increases with the ordinate. Note that while the computation of perfect solutions may be unfeasible for 256 nodes, already a 5-second heuristic search yielded results that reduced wait latency by 20 % compared to Chained Mode.

6.4 Summary and Conclusion

In this chapter, wait latency in periodic hypercube gossiping was analyzed and optimized. Combined results show that wait latency can constitute a considerable portion of total latency. Since they directly depend on the on the employed timing mode, overall latency can be reduced significantly. Five timing modes were investigated. Of these, the four local ones reduce wait latency by successively constraining transmission times at each node. Analytical bounds and expected values for wait latency and resulting distribution of total traversal time were provided and shown to yield accurate results. A fifth timing mode further improves upon the fourth by modeling and minimizing latency as a mixed integer linear program. Table 6.4.1 presents an overview of all findings.

6.4.1 Local Modes

Among the locally optimizing timing modes, particularly Chained Mode helps to significantly reduce worst-case traversal time. In the scenarios, wait latency was reduced by two thirds and overall latency was cut in half. The derived expectations and bounds are of enhanced practical value compared to simple maxima, let alone complexity bounds, and facilitate general statements as well as the selection of parameters without the need to run costly simulations.

The figures suggest that the latency predictions for the uniform network delay distribution also approximate the constant and even the life-like scenario reasonably well.

6.4.2 Global Optimization

For the timing scheme Chained Mode, the nature of wait latency in periodic hypercube gossiping was analyzed in more detail, and in Crossing Mode the problem of further minimization of wait latency was reduced to a mixed integer linear program.

Experiments attest for networks of 64 nodes an additional reduction of wait latency by close to 50 % on average compared to the best locally optimizing timing scheme Chained Mode. While for larger networks the effort for optimal solutions becomes prohibitive, still significant reductions can be obtained through heuristics within seconds.

Transforming the MILP problem to a completely integer one will allow to accommodate systems where input data only becomes available at k distinct phases and is left for future work.

So far, incomplete allocations and local queueing delay were not taken into account, i. e. all nodes were assumed to have separate uplinks for each neighbor. The evaluation of the implementations in the following chapters will use discrete event simulation to account for node-local uplink queueing, processing delay and incomplete and irregular hypercube allocations.

Table 6.4.1: Timing mode overview

Mode	Constraints	Wait Delays	Free Vars	$\hat{\Delta}$	$E[\Delta]$	M
Random	—	$\delta < T$	$2^L L$	$(L+1)T$	$\left(2^{L-1}L + 2^L - 1\right)\frac{T}{2(2^L-1)}$	$\sum_{h=1}^{L} \frac{\binom{L}{h}}{2^L-1} \Omega^{*(h+1)} * \chi^{*h}$
Sync	$\varphi_{n,\ell} = \theta_{n,-1}$	$\delta_{n,\ell,-1} = 0$	0	$L \cdot T$	$\left(2^{L-1}L - 1\right)\frac{T}{2(2^L-1)}$	$\sum_{h=1}^{L} \frac{\binom{L}{h}}{2^L-1} \Omega^{*h} * \chi^{*h}$
Spliced	$\varphi_{n,\ell} = \theta_{n,-1,0} + i \cdot T_{\text{slot}}$	$\delta_{n,\ell,-1} = 0,$ $\delta_{n,L,\ell} < T_{\text{slot}}$	$2^L(L-1)$	$(L - 1 + \frac{1}{k})T$	$\left(\left(\frac{L}{2} - 1 + \frac{1}{k}\right)2^L - 1\right)\frac{T}{2(2^L-1)}$	$\sum_{h=1}^{L} \frac{\binom{L}{h}}{2^L-1} \Omega^{*(h-1)} * \chi^{*h} * \omega$
Chained	$\varphi_{n,\ell+1} = \theta_{n,\ell} + \epsilon$	$\delta_{n,\ell+1,\ell} < T_{\text{slot}}$	2^L	$\left(\frac{L}{k} + \left\lfloor \frac{L-1}{2} \right\rfloor \left(1 - \frac{2}{k}\right)\right)T$	$\left(\left(2^{L-1}L - 1\right)\frac{1}{k} + \mathrm{g}\,(\mathrm{L})\left(1 - \frac{1}{\mathrm{k}}\right)\right)\frac{T}{2(2^L-1)}$	$\sum_{p=1}^{2^L-1} \frac{1}{2^L-1} \Omega^{*\mathrm{G}(p)} *$ $\omega^{*(\mathrm{H}(p)-\mathrm{G}(p))} * \chi^{*\mathrm{H}(p)}$
Crossing	$\sum w_e p_e =$ $\min \sum (w_e p_e)$	see Subsection 6.2.5	4	Wait latency is fully determined.		

Chapter 7

Implementation

Contents

A realization of periodic peer-to-peer hypercube gossiping aims to ascertain the practicability of the proposed approach. Although the implementation is intended as a functional prototype as opposed to a productive solution, a number requirements must be met:

- Field testing peer-to-peer applications comes with substantial effort, both in terms of resources and preparations. Also the reproduction of a realistic scenario poses a challenge. As a consequence, wherever possible, field tests are substituted with simulations. Clearly, the more use cases and code fragments can be covered by means of simulation, the less remain to be tested in costly field tests. This calls for a software architecture that minimizes the code portions that depend on the actual execution environment.
- Especially under simulation, the implementation should be highly efficient to allow for simulations with a large number of nodes.
- To facilitate experiments, the selection of timing modes should be possible through a configurable runtime parameter, i. e. without necessitating re-compilation. To allow for a continued support of a number of strategies, the realization of different timing behaviors should avoid code duplication.
- The implementation should be largely independent from specific data- and sensor-types, aggregation functions and use of the aggregation result.
- Implementation timing behaviors and peer-to-peer protocols are error-prone tasks. To facilitate separate verification of the function of the individual components, they should be unit-testable.

In this chapter, I present the prototypical realization of the approach established in Chapter 3, including four of the timing modes contributed in Chapter 6. The implementation

features a flexible and efficient C++ architecture and an extensible UDP protocol. The prototype can be run both on the OMNeT++ discrete event simulator [145] and as a stand-alone Linux application. Dependencies on the actual execution environment are encapsulated behind a facility interface. When run on the simulator, specific optimizations reduce the memory footprint and execution time of the application whereas supplemental diagnostic data is fed into the simulation framework. Flexible support for multiple strategies is achieved by applying the *mixin* pattern, harnessed for C++ in [47], using a combination of template programming and inheritance, thus avoiding code duplication. Data operations use templating to provide generic data handling and aggregation independent of data type. Unit-testability is ensured through loose coupling in conjunction with facility interface stubs.

7.1 Architecture

The realization of the periodic peer-to-peer hypercube gossiping approach is a peer-to-peer network in which each network node, called *peer* or *servent*, communicates with a small set of other peers, called his *neighbors*, by periodically exchanging UDP packets containing sequences of partially aggregated data along with some management information.

7.1.1 Layers

As shown in Subsection 3.4.2 on page 41, the overall aggregation scheme constitutes a directed acyclic graph (DAG) with each node forming a disjoint subgraph thereof.

Figure 3.4.4 on page 46 shows only the dissemination of information; the mode of performing the aggregation was left open. To this end, we introduce *accumulators* at each node, in between input and output nodes, as shown in Figure 7.1.1. A special meaning is attached to the edges leading from input nodes to accumulators. Thus, we consider four layers for the organization of the data processing architecture: An input layer, a delivery layer, an aggregation layer and an output layer.

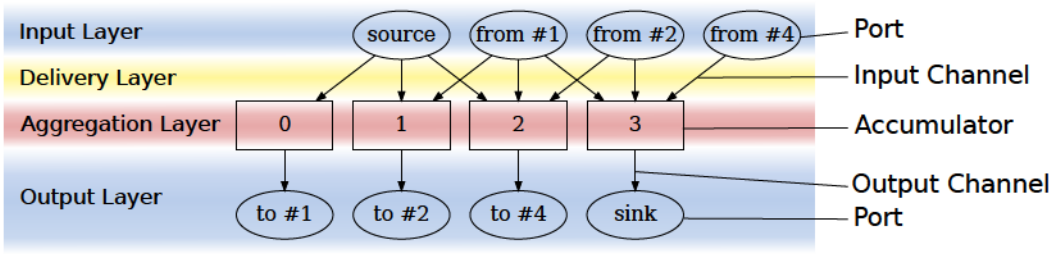


Figure 7.1.1: Data flow layers (node 0 of 8)

Input Layer

The input layer consists of so-called source *port* objects, one for each neighbor plus one for the local sensor, that receive, decode, and buffer data from the network socket or the sensor, respectively.

Delivery Layer

The delivery layer is constituted of *input channels* that receive data from the ports and feed it in a controlled way to the accumulators in the aggregation layer. It is responsible for jitter compensation and enforcing packet order.

Aggregation Layer

The aggregation layer is composed of a set of aggregation buffers, called *accumulators*, wherein data from different sources is aggregated.

Output Layer

The output layer includes *output channels* and the very same port objects as in the input layer, only this time in their role as output ports, that either encode data and send it to the neighbors or output it locally.

7.1.2 Components

Figure 7.1.2 provides an overview of the key components of the servent architecture. Aforementioned aggregation related components can be found on the right hand side. Below, the key components are discussed in detail.

Application

`Application` is an interface that encapsulates dependencies on the actual execution environment. It is implemented by class `RealApplication`, a stand-alone application that has access to a real sensor and network socket, and class `SimulatedApplication`, that integrates into the OMNeT++ simulation environment.

Servent

`ServentImpl` is the core component of the servent implementation. It can be accessed through the interfaces `ServentIf` and `ServentFacility`, providing external and internal views (*facets*) on the class's methods.

Messaging The messages exchanged between peers can be distinguished into two kinds: `DirectMessages` and `RoutedMessages`. The former are sent directly to a known socket address; the latter are routed hop-by-hop and piggy-back on routinely exchanged messages containing aggregation data.

Incoming UDP packets are handed to the `MessageFactory`, which parses the packets and extracts a number of `Message` command objects from each packet. These messages are passed to the `ProtocolHandler`, described below. Messages containing aggregation data (`DataMessages`) are handed to `Aggregator`, also described below. Outgoing `RoutedMessages` are queued at `Ports` for attachment to the next outgoing data packet. Outgoing `DirectMessages` are passed to `Application` for execution environment specific sending.

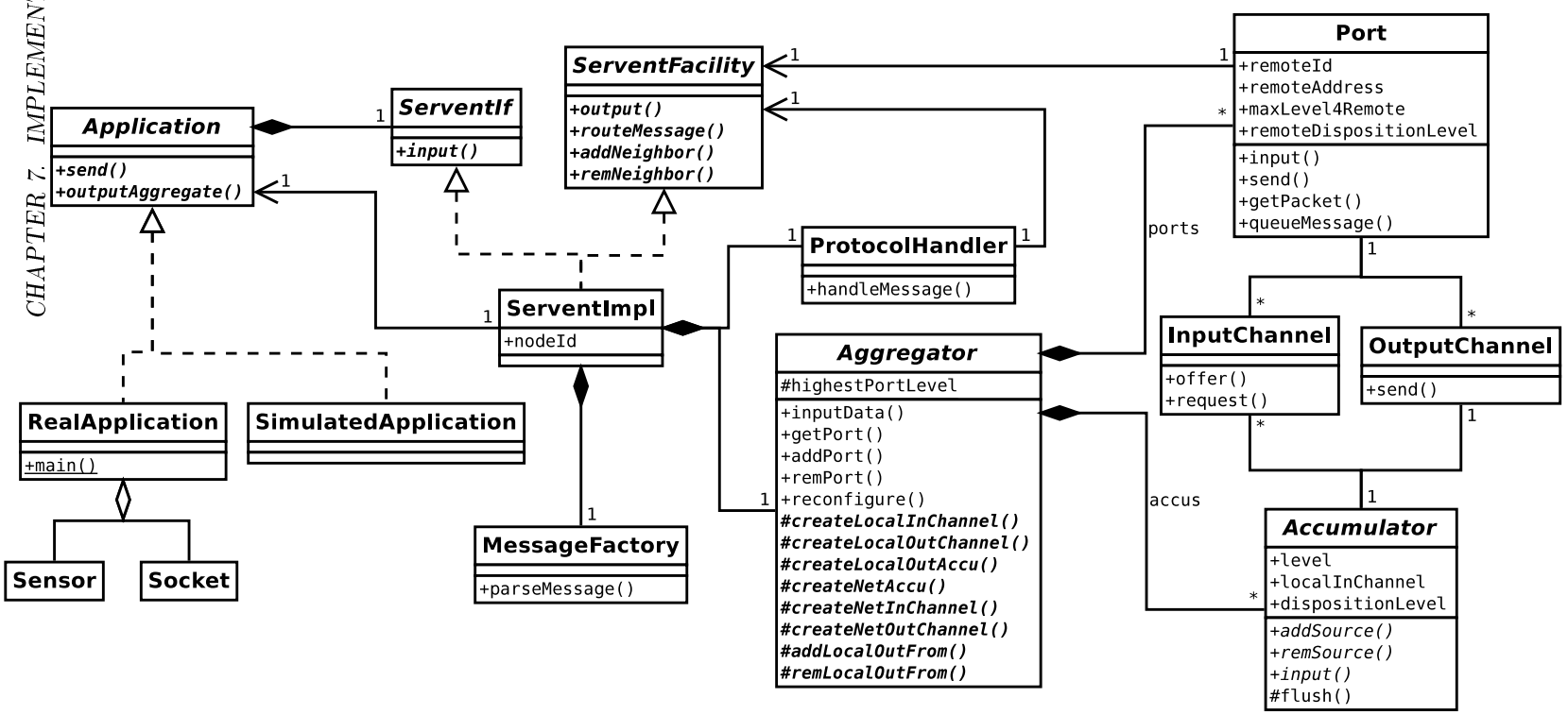


Figure 7.1.2: Servent architecture overview

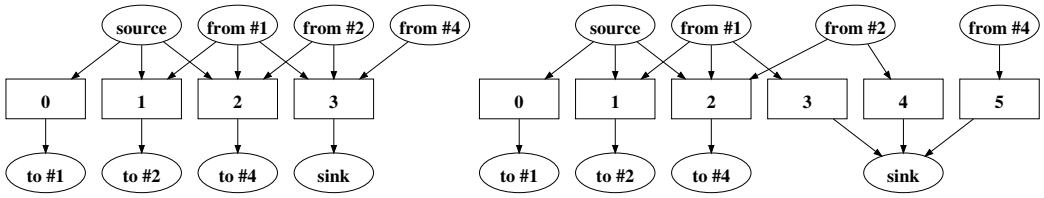
Memory Management To improve simulation performance, several optimizations are employed. When run under the simulator, message buffers use a global pool shared by all nodes. Packets are handed from source node to destination node without re-allocating the buffer or copying the message. Additionally messages are attributed with diagnostic data to ease debugging. This is achieved through the use of smart pointers and in-place instantiation of messages.

Protocol Handler

The `ProtocolHandler` processes messages to implement the peer-to-peer protocol, detailed in Section 7.2, and perform the network manipulations described in Subsection 3.3.4. At this, it forwards and creates messages and reconfigures the servent.

Aggregator

Data received from neighbors and the sensor is given to the `Aggregator`, that, in conjunction with `Ports`, `Channels` and `Accumulators`, implements the communication and aggregation scheme presented in Section 3.4 as well as the local timing modes described in Chapter 6. The different timing modes are realized by subclassing `Aggregator` and `Accumulator` as shown in Figure 7.1.4. They differ in how data from multiple neighbors



(a) `SingleLocalOutAggregator` (b) `MultiLocalOutAggregator`

Figure 7.1.3: Aggregation schemes for node 0 of 8 as used by Rand & Sync Mode (left) and Spliced & Chained Mode (right)

is fed to the data sink. In Random Mode and Sync Mode, one accumulator aggregates data from all neighbors, as shown in Figure 7.1.3(a). The accumulator is flushed once per period. In Spliced Mode and Chained Mode, data is fed to the sink right when it arrives. In this case, the sink is responsible for mixing the phase displaced data sources. This is shown in Figure 7.1.3(b).

When neighbors are added or removed or the maxlevel or summit (see Subsection 3.4.2 on page 44) of neighbors change, the aggregator needs to be reconfigured according to the periodic hypercube gossiping communication and dissemination schemes for the incomplete hypercube, as per Definitions 3.4.12 and 3.4.13, respectively. The imperative formulation of this reconfiguration is shown in Algorithm 7.1.

Accumulators

Accumulators are responsible for performing the intermediate, partial aggregation, and, in the cases of Random and Sync Mode, also the final aggregation. They can be thought of as data buffers that are initialized to the neutral element of the data type. Each input operation to the accumulator performs the fusion f of the contents and the new data,

Algorithm 7.1 Aggregator (re)configuration

```
// Create accumulators and wire destinations. Connections are always full-duplex.
complete = 0
for (l : 0..highestPortLevel) {
    dp = getPort(nb(l));
    // Create accumulators for connections to neighbors within the largest
    // complete containing hypercube and if either of us is the highest level
    // existing neighbor for the other
    if (dp && (complete==l || l==highestPortLevel || dp.maxLevel4Remote)) {
       if (!accus[l]) {
         accus[l] = createNetAccu(l);
         accus[l].localInChannel = createLocalInChannel();
         accus[l].destination    = createNetOutChannel(accus[l], nb(l));
       }
       accus[l].dispositionLevel = complete==l ? complete+1 : complete;
       if (complete==l && dp.remoteDispositionLevel==l+1)
         ++complete;
       addLocalOutFrom(dp);
    } else {
       accus[l] = null;
       if (dp)
         remLocalOutFrom(dp);
    }
}
// Wire sources for accumulators
for (acc : accus) {
    dp = acc.destination.port;
    for (a : accus) {
       sp = a.destination.port;
       if (sp == dp)
         continue; // Never return data to same neighbor
       // Disseminate each source to all levels >= source's disposition level
       if (sp.remoteDispositionLevel <= acc.level)
         acc.addSource(sp.remoteId, createNetInChannel(sp, acc));
       else
         acc.remSource(sp.remoteId);
    }
}
```

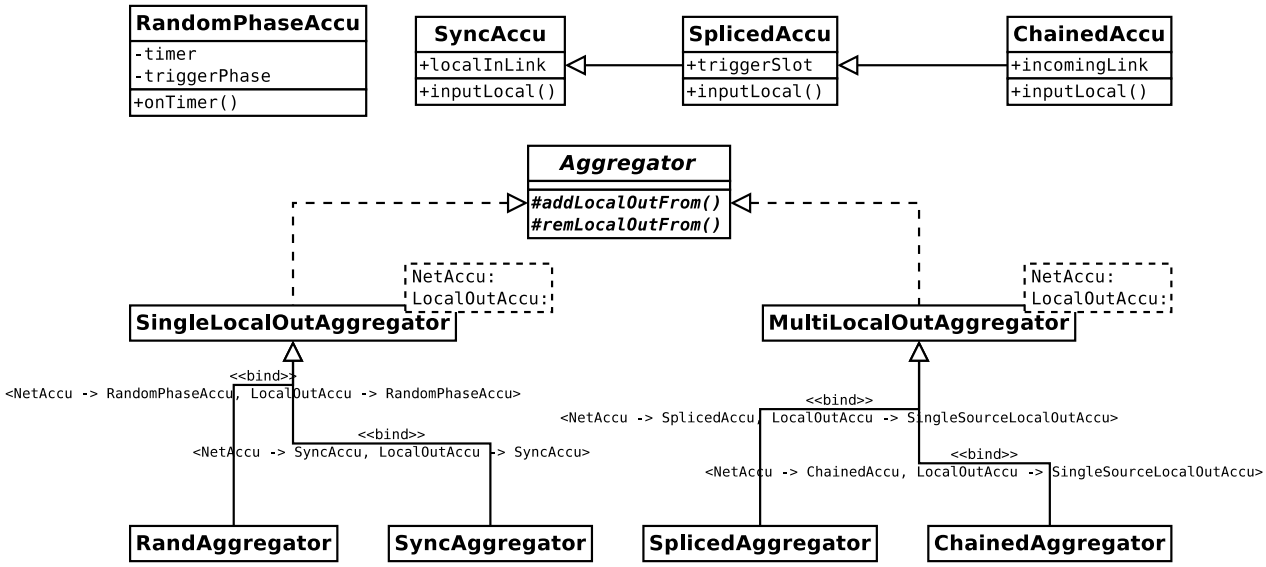


Figure 7.1.4: Realization of timing modes through aggregators

according to Section 3.4. The result of the fusion is again stored to the buffer. At some point, specific to the timing mode of the accumulator, it is flushed, i. e. its contents are committed to the attached `OutputChannel` and the buffer is re-initialized. According to the timing mode, each accumulator computes the *due time*, i. e. the time at which the next flushing will occur and requests from all `InputChannels` the latest data that will have arrived before said due time. When the due time will have arrived, it will request from all input channels not having delivered an extrapolated packet, as discussed in the next subsection.

The accumulators associated with the different timing modes vary mostly in their method to trigger the flushing. `RandomPhaseAccu` uses a timer initially set to a random phase. `SyncAccu` uses the arrival of local sensor data every period. So does `SplicedAccu`, however the slot may be different for every accumulator of the node. `ChainedAccu` mode uses the slot arrival of local sensor data right after the estimated arrival of the preceding chain link.

Ports

Ports receive data packets from the network socket or the sensor, on the one hand side, and from output channels, on the other hand side. They perform the following tasks:

Arrival Estimation The reception times are measured and a high probability quantile of the reception phase is computed by taking into account the past arrival times and changes in send phase, specifically by computing a quantile of the time difference between the remote peer sending a packet at remote time and this peer receiving it at local time. This way, source ports are able to provide time estimates by which future packets will have arrived with a high probability.

RTT Computation Round trip time *(RTT)* to the remote peer is continuously measured with piggy-backed acknowledgements whilst deducting the remote sojourn time and adding the times by which the packets arrived early with regard to $\chi_{n_1,n_2,q}$.

Congestion Detection When one node experiences rising RTT values to all neighbors, this is strong evidence for the congestion of that node's Internet access link[1]. In this case, it will terminate its connections and request a bandwidth reduced position. Note that packet loss, though often pointed out, is ill-suited as a congestion indicator in peer-to-peer scenarios because DSL or cable modems can queue several seconds worth of packets before packet loss occurs.

Decompression After receiving and measuring the packet, the source port prepares the received data for subsequent use, e. g. converts it from transfer format into a format suitable for aggregation.

Buffering The packet is then stored to a ring buffer. This buffering does not introduce extra latency. In fact, packets are only queued for the amount they arrive early with regard to their ETA $\chi_{n_1,n_2,q}$ plus wait delay described in Chapter 6.

Notification Channels that have registered interest are notified of the arrival.

Extrapolation / Loss Concealment For some applications, in the case of lost packets, it makes sense to rather provide extrapolated data than no data at all. For example, in an audio stream, missing samples can be synthesized. The extrapolated data must be available when the packet is due and has not arrived. Consequently, the synthesized packet must be computed ahead of time, e. g. for the next packet, directly after notifying the connected channels of the current arrival.

Message Forwarding Ports accept `RoutedMessages` from `ProtocolHandler` for piggy-back routing. They store these messages in a FIFO queue until the next data transmission.

Compression and Sending Before the sending, data may have to be converted to transfer encoding. Finally, it is passed through the `Application` interface to the execution environment for transmission.

Channels

Input channels deliver data packets in a controlled way from source ports to mixing buffers. On their way from one node to another, packets can get delayed, lost, or even re-ordered. Input channels, in conjunction with source ports, provide an abstraction that hides away the unfavorable irregular characteristics of the Internet. A channel's client, i. e. `Accumulator`, can be sure of the following:

- Packets will be delivered in monotonic order and no packet will be delivered twice.

[1] Most likely, it will be the node's uplink, which is generally narrower. While the directed path on which the bottleneck lies could be determined from the timing information at hand, in our case it does not matter; it is to sufficient to know that either upstream or downstream bandwidth is scant.

- Each packet will be delivered only after having been requested.
- Packets will be delivered either before the specified deadline or not at all.
- With specified high probability, some packet will be delivered in time but packets will not be buffered longer than necessary to ensure this.

To this end, channels keep track of reservations and the sequence numbers of the last delivered packets.

Delivery Figure 7.1.5 illustrates the interplay between ports, channels and accumulators by example of accumulator 1 of node 0 of 8 as of Figure 7.1.1. The depicted order of events could occur in any of the Sync, Spliced and Chained mode.

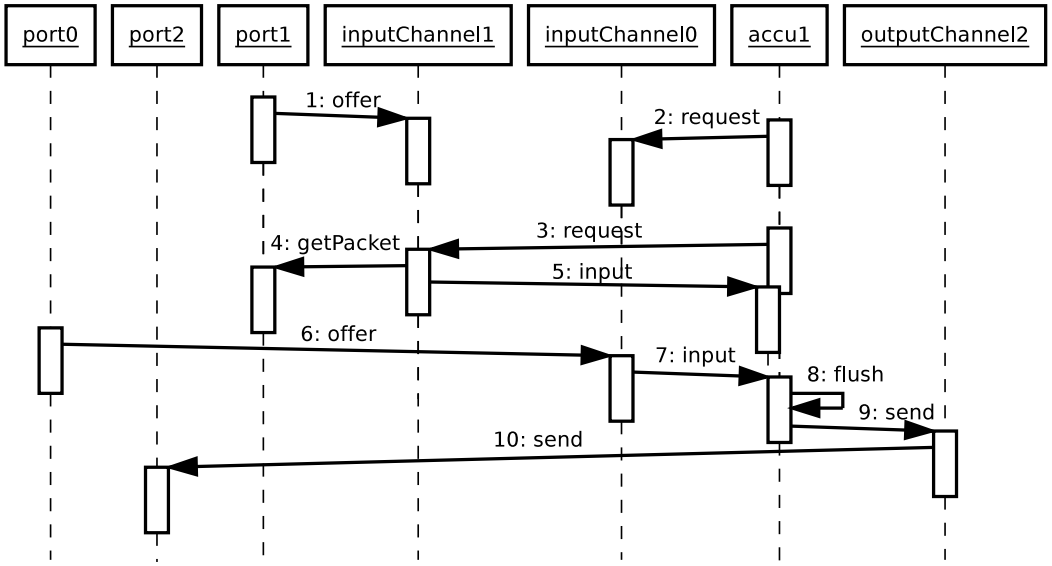


Figure 7.1.5: Servent data flow sequence

1. `port1` notifies `inputChannel1` of the arrival of a new, numbered packet. However, the channel's receiver has not yet requested a packet, so the channel just notes the sequence number of the arrival.
2. `accu1` request a packet from `inputChannel0`. Since this channel is its `localInChannel`, connected to the local sensor, `accu1` request the data to be delivered *after* t_0. The `InputChannel` computes the sequence number of the target packet matching the requested time. Since the packet has not yet arrived, the channel just notes the request and returns the estimated time of arrival (*ETA*) t_1.
3. `accu1` requests from `inputChannel1` a packet to be delivered before t_1, i. e. the ETA returned by `inputChannel1`. Using the associated port's arrival estimate, `InputChannel1` computes the sequence number of the target packet matching the requested due time.
4. Since the target packet has arrived already, the channel retrieves it from the associated port's ring buffer.
5. The retrieved packet is handed to `accu1`.
6. Fresh data from the sensor is offered to `inputChannel0`.
7. Because of the registered request, it is passed right through to `accu1`.
8. `accu1` fuses the data with its current contents. Since the arrival was from the accumulator's `localInChannel`, it also triggers flushing.
9. `accu1` outputs the partial aggregate.
10. `OutputChannel2` just passes the data through to the port from where it is sent to node 2.

7.2 Protocol

The servents exchange information using UDP datagrams. In contrast to TCP, UDP is unreliable and provides no congestion control. This makes it especially suited for real-time applications. For these, transmission problems are best handled on the application layer. For instance, commonly there is no use in resending dropped packets as they would likely arrive too late to be of any value, and moreover, generate excess traffic that may even compound the problem.

To enable deployment in typical peer-to-peer scenarios, the protocol must consider that hosts may be hidden behind firewalls and NAT routers, as particularized in Subsection 2.4.5. The network setup is performed as described in Subsection 3.3.4.

As each sent packet comes with an overhead, the protocol aims to attach management information to the aggregation data, which is exchanged periodically, where possible. Messages can thus be discerned into *direct messages*, that are sent directly from one node to another, and *routed messages*, which travel piggy-back with the aggregation data.

Inside the UDP payload, the data is organized in a flexible container format based on the Electronic Arts Interchange File Format [112], whereat the payload is divided into variable size portions, called *chunks*. Each chunk begins with a four-byte chunk identifier, corresponding to a sequence of printable ASCII characters, called the type ID or FourCC. The type ID is followed by a four byte chunk length, not counting the type ID and chunk length fields. Next comes the actual chunk content. To make the whole chunk a multiple of 4 bytes, it is followed by 0–3 padding bytes. This scheme is illustrated in Figure 7.2.1. Employing this simple container format has several benefits:

- An arbitrary number of logical messages can be embedded in one packet.
- The individual parts can be of variable size.
- The protocol can be easily extended by simply adding further chunks.
- Parsing is very efficient.
- Using a FourCC as chunk identifier eases debugging, as the four character string can be spotted easily in packet dumps.

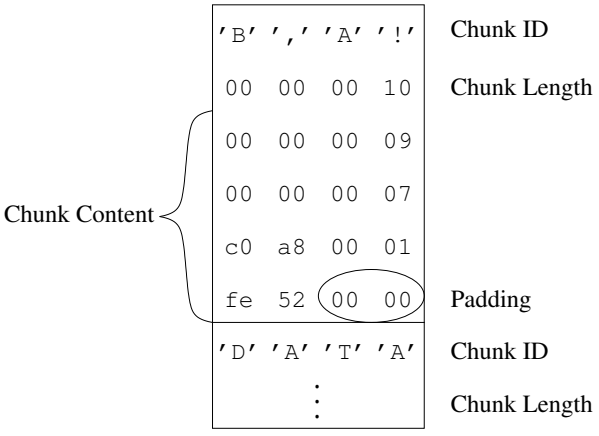


Figure 7.2.1: Message organization within UDP payload

Eleven message chunks are currently defined and will be described below. Throughout the description, the following terms will be used:

applicant node applying for a node ID

contact node receiving applicant's request

host node accepting applicant as a neighbor

An exemplary message exchange is shown in Figure 3.3.5 on page 32.

FourCC:	`RQID`	Name:	**Request ID**	Type: direct
Synopsis:	Application for node ID.			
Content:	`uint16_t maxNeighbors;`			
Description:	Sent by unassociated node to any node of the network. The receiver (*contact*) will either reply directly with PTMY or will forward the request in the form of ACMD in the direction of a prospective host. `maxNeighbors` is used to determine whether the host is bandwidth-impaired and requires special placement.			

FourCC:	`ACMD`	Name:	**Accommodate**	Type: routed
Synopsis:	Forwarded request for node ID.			
Content:	`source_t`	`contactId;`		
	`Inet4SocketAddress`	`applicantAdr;`		
	`uint16_t`	`maxNeighbors;`		
Description:	Host will reply with HOST, routed back to contact, and PTMY, sent directly to applicant; otherwise this message is forwarded to next prospective host.			

FourCC:	`HOST`	Name:	**Found Host**	Type: routed
Synopsis:	Positive reply for ACMD.			
Content:	`source_t`	`hostId;`		
	`source_t`	`contactId;`		
	`Inet4SocketAddress`	`applicantAdr;`		
	`Inet4SocketAddress`	`hostAdr;`		
	`level_t`	`level;`		
Description:	Routed back to contact, who will send RFR2 to applicant.			

FourCC:	`RFR2`	Name:	**Refer To**	Type: direct
Synopsis:	Replied by contact to applicant upon received HOST.			
Content:	`Inet4SocketAddress neighborAddress;`			
Description:	Applicant is to re-send RQID to supplied address. Since host has sent PTMY already, the RQID message completes the hole punching.			

FourCC:	`PTMY`	Name:	**Pleased to meet you**	Type: direct
Synopsis:	Receiver is accepted as sender's neighbor.			
Content:	`source_t myId;`			
	`source_t yourId;`			
Description:	Sent by host to applicant either on RQID or ACMD or B,A!.			

FourCC:	`B,A!`	Name:	**Introduce A**	Type: routed
Synopsis:	Routed by host to potential neighbors of applicant.			
Content:	`source_t`	`aId;`		
	`source_t`	`bId;`		
	`Inet4SocketAddress`	`aAdr;`		
Description:	The receiver will reply with A,B! (routed) and PTMY (direct).			

FourCC:	`A,B!`	Name:	**Introduce B**	Type: routed
Synopsis:	Returned socket address of B.			
Content:	`source_t`	`aId;`		
	`source_t`	`bId;`		
	`Inet4SocketAddress`	`bAdr;`		
Description:	Routed back upon reception of B,A!			

FourCC:	`BYE!`	Name:	**Good-bye**	Type: direct
Synopsis:	Politely sent to all neighbors upon termination.			
Content:	`source_t myId;`			
Description:	Tells receiver that he may immediately commence reconfiguration without waiting for the timeout to pass.			

FourCC:	`DUEL`	Name:	**ResolveConflict**	Type: direct
Synopsis:	Informs holders of the same ID of each other.			
Content:	`source_t`	`id;`		
	`Inet4SocketAddress`	`address;`		
	`uint16_t`	`key;`		
Description:	Recipients will DARE each other. `key` uniquely identifies this fight.			

FourCC:	`DARE`	Name:	**Challenge**	Type: direct
Synopsis:	Challenge receiver for a fight for id.			
Content:	`uint16_t key;`			
	`source_t id;`			
	`source_t nodeCount;`			
	`struct { source_t id;`			
	`Inet4SocketAddress address; } neighbors[];`			
Description:	The node with higher `nodeCount` wins this fight identified by `key`. The passed neighbors are examined for further ID conflicts.			

FourCC:	`DATA` Name: **Aggregation Data** Type: direct
Synopsis:	Partial aggregate data exchanged between neighbors.

Content:	`source_t sender;` // *same as* `PTMY.myId`
	`source_t recipient;` // *same as* `PTMY.yourId`
	`seqno_t pktseqno;` // *arrival estimation*
	`time_t sdt;` // *arrival estimation*
	`seqno_t ack;` // *rtt computation*
	`time_t ackDelay;` // *rtt computation*
	`level_t dispositionLevel;` // *see Subsection 3.4.2*
	`level_t maxLevel;` // *see Subsection 3.4.2*
	`source_t nodeCount;` // *aggregate*
	`source_t maxId;` // *aggregate*
	`holes_t holes;` // *aggregate, see Subsection 3.3.4*
	`source_t nsources;` // *aggregate*
	`data_t data;` // *aggregate*

Description:	Beside the actual data, four more aggregates are formed for internal purposes.

7.3 Limitations

In this chapter I have outlined the realization of the approaches of Chapters 3 and 6 in a generic, application independent, form. To this end, I have picked a small and somewhat arbitrary selection of hopefully interesting implementation details out of 20 k lines of code.

To turn the presented prototype into a useful application, several more issues would have to be addressed:

- Currently, aggregation is limited to a single session.
- IPv6 is not supported.
- The implementation supports only binary hypercubes.
- Only Random, Sync, Spliced and Chained Mode are implemented.
- To make better use of available computing resources, the application should use multiple threads.
- To ease employment in varied applications, the stand-alone program should be transformed into a generic aggregation framework.

Chapter 8

Application 1: Scalable Audio Communication for Massively Multiuser Virtual Environments

Contents

With the advent of virtual environments and MMORPGs came the desire to not only hear the sounds of the environment but also talk naturally, that is, with an audio model conforming to the virtual environment. Audio communication for massively multiuser virtual environments (MMVEs) comprises the provisioning of hundreds of thousands of users with their personal, position dependent compositions of audio streams, in real-time, at low latency, using non-realtime components connected through a heterogeneous network with diversely limited bandwidths and varying delays.

The first audio communication solutions for virtual environments were separate programs that disregarded virtual distance, orientation, room acoustics, and 3-D sound altogether. Recently, there have been various efforts to integrate audio communication into the virtual worlds. These however do not perform satisfactorily in the context of dense

crowds. In lack of functioning comprehensive solutions, users resort to detached and limited conferencing tools.

In this chapter, I will show how the periodic peer-to-peer hypercube gossiping approach, presented in Chapter 3, can be applied to facilitate scalable audio communication for MMVEs. To this end, the base architecture, described in Chapter 7, needs to be adapted to suit the application specific circumstances and requirements. A particular problem regarded comprises the mapping of virtual scenes to hypercube allocations. The contributions of Chapter 6 are employed to reduce and to quantify latency. The prototype is test-driven using OMNeT++ simulator in conjunction with the DELFOI delay model from Chapter 4. Latency results indicate feasibility for domestic scenarios.

The rest of this chapter is organized as follows: Section 8.1 motivates the application use case and defines the problem based on a brief on audio communications in virtual environments. Section 8.2 gives an overview of related work. Section 8.3 regards needed modifications to the aggregation scheme as well as the problem of mapping virtual scenes to hypercube allocations. Section 8.4 deals with technical considerations for the implementation. In Section 8.5, the system's performance is evaluated through simulation, and Section 8.6 concludes this chapter.

8.1 Background and Problem Definition

With the rise of a whole new generation of massively multiplayer online games *(MMOGs),* such as Lineage II, World of Tanks, EVE Online, and Second Life, MMVEs are now experiencing hundreds of thousands of concurrent users with an upward trend. Their operation long since requires thousands of servers, each [23, 142, 89]. At the same time, MMVEs are becoming increasingly interactive where interaction more and more resembles natural conversation [133]. In *auditory virtual environments (AVEs)* [116] all participants are allowed to speak at once, but each one only hears a subset of the others, possibly with different audio volumes depending on their distance. Although some of the games provide audio systems already, these still have problems coping with dense crowds created by large numbers of users gathered at the same virtual location, e. g. attending the same events or activities such as New Years Eve's countdown or popular sports events. Unsatisfied users resort to using external tools such as TeamSpeak, Mumble or Skype for "critical" communication. By shifting the load to the abundant clients, peer-to-peer communication promises to provide better service quality at lower cost.

Technically, audio conferencing is a special manifestation of the gossip problem, described in Subsection 2.1.3, in which every node's information must be communicated to all other nodes. However, for audio conferencing in virtual environments, the final result is not just the aggregation of all sources. In particular, each participant needs a composition of the other participants' audio streams, called *personal mix,* at which each audio source is scaled by a factor that may be different for each listener.

8.1.1 Conventional Audio Conferencing

Conventional audio conferencing can be regarded as a special case of sensor fusion, where the sensors are microphones, the streams are series of audio samples and the aggregation function is the sum over all sources. Each participant needs a composition of many other participants' audio streams, e. g. all audio streams except her own.

The first conferencing solution was *telephone conferencing*, dating back to the beginning of the twentieth century [122]. Conference providers used analog circuits for superpositioning the audio signals, which until the 1980s were continuous in range and time.

By contrast, in digital telephony, such as ISDN [70, 71] or GSM [46], an analog-to-digital converter (*ADC*) converts the continuous signal into a discrete time digital representation. The samples are encoded and streamed across the network to the conference provider, who combines audio streams by a process called *mixing*, which involves digitally summing the corresponding audio samples from the incoming streams and then normalizing the result [126]. This technique also applies to *voice-over-IP (VoIP)*, where IP-packets are used to transfer audio data. The transmission of each packet induces an data overhead due to various headers as well as associated processing effort for labeling and checksumming. Sending each audio sample in its own packet would generally overload both the processing node and the network link. As a consequence, audio data is not sent as separate samples but in chunks of many consecutive samples. This chunking also facilitates the use of more efficient compression, such as quantization in the frequency domain or, for instance, with the aid of psycho-acoustic models, but causes additional latency.

Latency in Communication In some streaming applications, the timeliness of the delivery matters, however, not in the notorious ones. In unidirectional streaming of previously recorded audio or video, such as music or movies, latency plays a secondary role. Delay variations known as *jitter* can be eliminated by use of generously dimensioned buffers on the receiving side. But even so-called "live" streaming usually has only moderate latency requirements. Latency only becomes a nuisance when information arrives faster by a side channel, e. g. when the viewer of a live video stream hears the cheering of fans coming from the stadium before he sees the goal happen on his receiver. In real-time voice communication, however, excessive latency becomes a real problem as it adversely affects conversation in various ways:

- The perceived annoyance by line echo increases with the delay the echo is subjected to.
- In a phone conversation the media access control, i. e. who gets to speak, is negotiated using a CSMA/CD-like "protocol" similar to the one employed in Ethernet. A (polite) participant wishing to say something, first waits for the line to be silent, then starts speaking. If both participants started speaking at the same time they will notice and stop. In a domestic phone conversation this takes place very quickly. If the delay is increased, however, the collision can only be detected at a much later time, where both participants have assumed the channel to be clear, which makes it even more unpleasant to stop, e. g. in mid-sentence. As a consequence the speakers automatically resort to a form of CSMA/CARP and before starting to speak not only wait for silence but also wait some additional period to make sure the previous speaker had really finished speaking.
- When one participants asks another a question, the spoken reply arrives a full round-trip-time later than it would if both speakers were in one place. Even worse, additional delay is added by the CSMA/CARP collision avoidance mentioned above. The resulting unfamiliar pauses make conversations tiresome and increase the perceived distance between the conversation partners.

For these reasons, latency in audio communication should be kept to a minimum.

8.1.2 Auditory Virtual Environments

The intent of auditory virtual environments is to create a context in which participants have auditory perceptions that correspond not to their real but to their virtual surroundings [15]. The goal is a very plausible or even natural reproduction. The participant is to gain a spatial sensation of his virtual environment and perceive both his own movements in the surroundings as well as the movements of the other sound sources.

Whereas in some applications the user is just a passive receiver of his environment, we will regard the more challenging use case where users communicate with each other in real-time. At this, communication not only includes direct conversations but also all nuances in between loosely overhearing a conversation at the neighboring café table to the atmosphere-creating ambient buzz. Since audio communication also includes spontaneous, nonverbal vocal utterances such as laughs and sighs, it is obvious that solutions with push-to-talk and similar floor control can be ruled out.

Scene Accuracy & User Expectations Just as in a real environment the auditory perception is determined by the acoustics of the surroundings, in an auditory virtual environment the auditory perception is determined by the audio model of the virtual environment. At this, however, it is not sufficient to reproduce the acoustics on a mere physical level. The participant's sensation is also influenced by listening experience, attention and expectations [16].

One must also consider that the perfect auditory experience may not be achievable for a high number of users with limited bandwidth and computational resources. We will therefore strive for a scalable solution that is as convincing as possible. Perfect scene accuracy between participants can only be achieved with separate mixing operations whereat each audio source is scaled by a factor that may be different for each listener. This means that the perfect result is a *holistic aggregate* (see Section 2.3), whereat no useful partial aggregation can be done, and all the data must be brought together to be aggregated by the evaluator. Clearly, transporting all streams in unaggregated form to all nodes for local mixing is not feasible for lack of bandwidth and computational power. In some cases, however, an approximation of the scaling factors may be sufficient so that some mixing results can be shared by several listeners.

Latency For auditory virtual environments the problem with latency in communication is even exacerbated. [133] explicates the need for latency bounds as follows:

> "One characteristic of MMVEs is that they are very interactive. Users can almost act as if they are seeing or talking to each other in a face-to-face manner. Thus, delay among users, typically caused by latencies on their communication links, must be bounded. The actual range depends on the application type, but is often in the range of a few hundreds, or even a few tens of milliseconds."

In some games, teams not only need to discuss strategy but also issue tactical orders in action [132]. This raises the bar with respect to latency requirements even higher.

8.1.3 Problem Definition

In summary, the key challenge in massively multiuser audio communications is the provisioning of participants with personal position dependent compositions of live audio streams

(mixes) of audio data with acceptable latency, scene realism and packet loss rates using non-realtime components connected through a heterogeneous network with diversely limited bandwidths and varying delays. To meet user expectations, the solution should provide adaptive quality depending on virtual locality: Especially for users standing virtually close together, low latency and high scene accuracy should be aspired.

8.2 Related Work

Diverse architectures have been considered for audio delivery in conventional conferences as well as virtual environments. They differ greatly in their assumptions regarding network topology, features, and available bandwidths. Table 8.2.1 compares the resource demands of a selection shown in Figure 8.2.1 and described below.

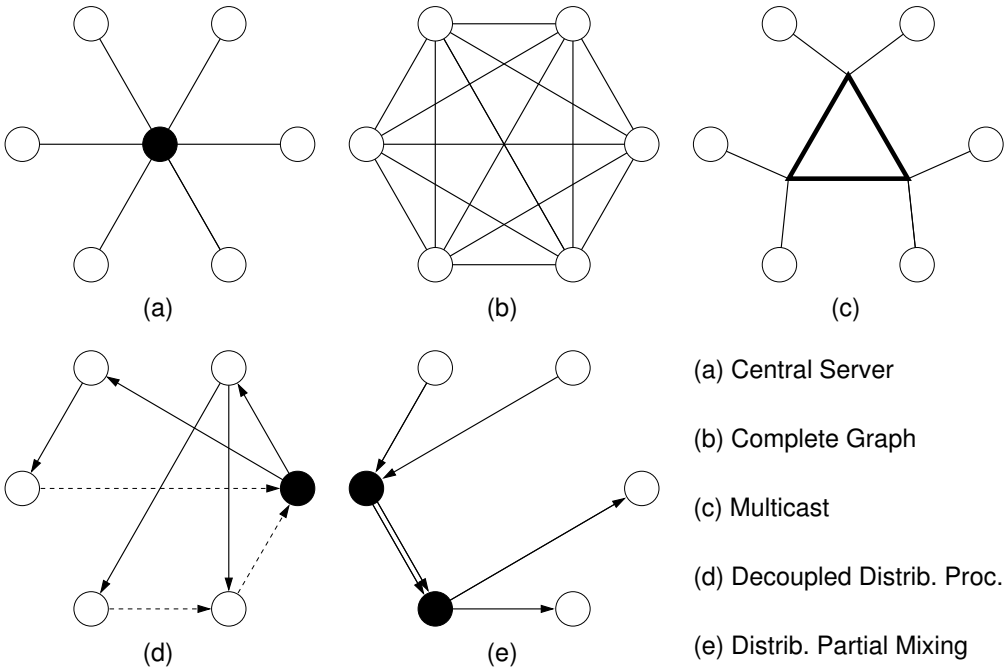


Figure 8.2.1: Selection of distribution topologies

Central Server In a central server setup, i. e. *star* topology (a), all participants send their audio data to a server that mixes the streams and sends them back to each client. With respect to the number of users N, the provider's traffic grows in the order of N and the processing effort in the order of N^2, thus making scalability costly for the provider. Nevertheless, the approach is viable for small conferences or where revenues pay for server operation and traffic. One notable advantage of the central server approach is that—sufficient server bandwidth provided—it allows for fully individual mixes and low latency with minimal client bandwidth. It is used by the conferencing solutions TeamSpeak, Ventrilo, Mumble, and Skype.

Complete Graph The complete graph topology (b), proposed in [93, 76] stipulates that each participant sends his data to every other participant, thus obviating the need for a dedicated server. While the complete graph is the most flexible topology, it also has the highest bandwidth requirements. Each node's processing effort and bandwidth are in the order of N. As described in [93], such structure works well for small-to-medium size conferences but is less practical for bandwidth-limited end systems such as users with asymmetric DSL connections with low upstream bandwidth and does not scale well to larger conferences.

In VOISCAPE [76] a multi-context voice communication system called is described where "users can talk with other users and move, in a way similar to face-to-face conversation, in a virtual auditory space". The prototypical voiscape implementation uses peer-to-peer real-time communication. Whenever two avatars move within hearing range of each other a new bi-directional SIP/RTP connection is negotiated. This leads to a fully-meshed communication structure, i. e. complete graph.

[94] proposes a novel distributed mixing approach in which super peers form a fully connected graph and then take turns to mix audio and send it to the other participants. While averaged over time, this indeed reduces the traffic to constant and the computation to linear effort for each node, it does not, however, improve latency.

Multicast For the—usually more critical—upstream direction, the bandwidth problem may be alleviated by use of multicasting or proxy servers—if available [21]. The multicast topology (c) saves peer upstream bandwidth by replicating streams over a multicast backbone, in the figure denoted by the thick triangle. However, multicast is neither well supported by Internet nor widely deployed and rarely available to end users [118].

Decoupled Distributed Processing True remedy lies in the spatial distribution of the mixing process, by hierarchical mixing. Instead of delivering every source by itself, the audio streams can be mixed in several stages on intermediate nodes, thereby reducing data volume on the way. In peer-to-peer stream mixing, the mixing is performed by the client nodes themselves so that no servers are required. With PEERTALK [58] a resource-efficient two-phased structure, called *decoupled distributed processing (DDP)*, is presented, whereby the audio stream processing is decoupled into an *aggregation phase* that mixes the audio streams of all active speakers into a single stream by way of a mixing tree, and a *distribution phase* that distributes the mixed audio stream to all listeners via a distribution tree (d). Whilst a valid solution for regular conferencing, this approach is not suited for AVEs. Because the voices of all participants are concentrated on one node, called the *root mixer*, it is not possible to provide a personal mix to each participant. Also, the delay associated with one participant's reception of the other participants' streams is determined by his position in the distribution tree.

Distributed Partial Mixing [126] deals with the problem of providing personal audio composition by an approach called *distributed partial mixing (DPM)* that comprises combining only some of the streams and transporting other streams separately where bandwidth permits (e). Their single tree approach, however, appears to be aimed at scenarios with peers behind mixer network components at network boundaries, e. g. corporate gateways or dial-up servers rather than mesh networks like the Internet and individual peers.

See [21] for an extended comparison of delivery architectures for immersive audio in crowded networked games.

Table 8.2.1: Topology comparison

Effort	Star	K_n	Multicast	DDP	DPM	H_m
server bandwidth	max	–	–	–	high	–
client/peer bw up	low	high	low	medium	low	medium
client/peer bw down	low	high	high	medium	variable	medium
server mixing effort	high	–	–	–	variable	–
client/peer mixing effort	none	high	high	low	variable	medium
individual channel control	yes	yes	yes	no	variable	somewhat
latency	(low)	(low)	low	medium	medium	variable
special nodes	server	none	none	root mixer	partial mixers	none

8.3 Mapping the Virtual Environment to the Hypercube

Audio communication in MMVEs requires a system that provides a personal mix of audio sources for each participant at low latency and bandwidth and is scalable with regard to the number of participants; a system that is a low-complexity, low-degree solution to the gossip problem in full-duplex mode. The periodic peer-to-peer hypercube gossiping approach, presented in Chapter 3 satisfies these requirements. To save bandwidth by in-network-aggregation, the aggregate is required to be distributive (see Section 2.3). However, the requirement of perfect personal mixes turns the aggregate into a holistic one. The proposed solution to this dilemma is to treat the aggregate as distributive but minimize the concomitant error by a considerate mapping of avatar positions to node IDs. This mapping strategy will be purport of this section.

8.3.1 Continuity

In the past, MMOGs had distinct locations or rooms and each user was located in one room or another. Now, MMVEs develop towards continuous spaces wherein users can move seamlessly. Despite the good scalability of the hypercube topology, putting all users into one hypercube network is neither feasible nor necessary. Instead, the virtual space should be partitioned. In (floored) indoor environments, as well as in most outdoor MMVEs, where users linger on some (e. g. planetary) surface, it is sufficient to partition the surface into two-dimensional cells. Distance is best preserved using a hexagonal grid, also proposed in [152], and shown in Figure 8.3.1. For any pair of adjacent cells at most one node of the first cell communicates with at most one node of the second cell to exchange cell aggregates. If cells are smaller than the hearing range, it is still possible to convey audio across several cells along the three axes but not in between, as this would result in redundant data paths, and thus, echoes[1].

Only in environments with multi-storied open structures where users move freely through the third dimension, 3-D cells, e. g. *hexagonal close-packed (hcp)* or *face-centered cubic (fcc)* are suitable.

In environments where avatars move or act as groups, it may be preferable to use moving cells to facilitate coherent communication within the group. The predominant use of external communication tools suggests that this is likely more important than high fidelity with respect to physical acoustics.

[1] For some environments this effect could actually be acceptable or even desirable.

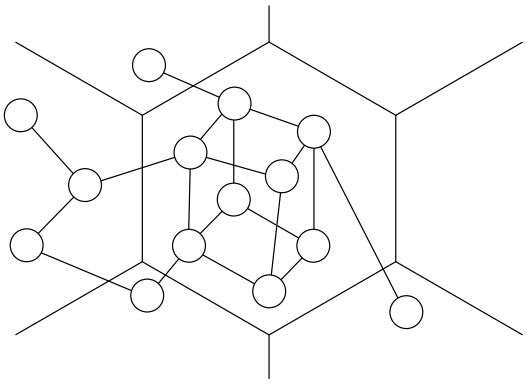


Figure 8.3.1: Nodes at avatar positions in hexagonal cell

8.3.2 Adapted Aggregation Scheme

In Figure 3.4.3 on page 45 a generic hypercube aggregation scheme was shown. The aggregate was formed from all sources. For audio conferencing, in contrast, aggregates are required that exclude each participant's own audio. An aggregation scheme that is suitable for audio communication is shown in Figure 8.3.2. The final result, displayed in the rightmost column, does not contain the original piece of information of the respective node, i. e. corresponds to (3.4.6). The different shades in the rightmost column represent the attenuated volumes of the personal mixes. Obviously, the received pre-mixed partial aggregates can only be scaled as a whole, so the final mixing result deviates from the perfect personal mix that would be attainable if all sources were separately available at each node. The mapping algorithm presented below aims to assign node IDs in such a way as to minimize this error.

8.3.3 Mapping Algorithm

A mapping algorithm is needed to transform the virtual avatar locations into node positions in the network. At this, the resulting latency distribution and fidelity of each participant's personal mix must be considered. Furthermore, the algorithm must be stable to avoid drastic dynamic changes of cluster memberships. In order to achieve a perfectly accurate rendering of the scene according to the sound model (*perfect scene accuracy*), for each listener the audio volume of each speaker needs to be attenuated according to the distance between them. Because of the proposed processing structure, where mixed subsets of streams are shared, this is not possible. For its output each peer can only attenuate the composite streams it receives. The higher the level on which a composite stream is received, the more sources it contains (and are scaled by the same factor) and the coarser is the granularity. As latency grows with the path length, a high hop-count should correspond to a high virtual distance.

Even though perfect scene accuracy is not attainable, the algorithm should perform the clustering in an intuitive way, so that the participants' expectations (e. g. who will hear them) are approximated as closely as possible. Especially two participants standing closest to each other should experience a low latency between them and hear each other loud and clear and with a high scene accuracy. It is desired that participants standing further away should be heard with lower volumes. Because they blend with the background noise,

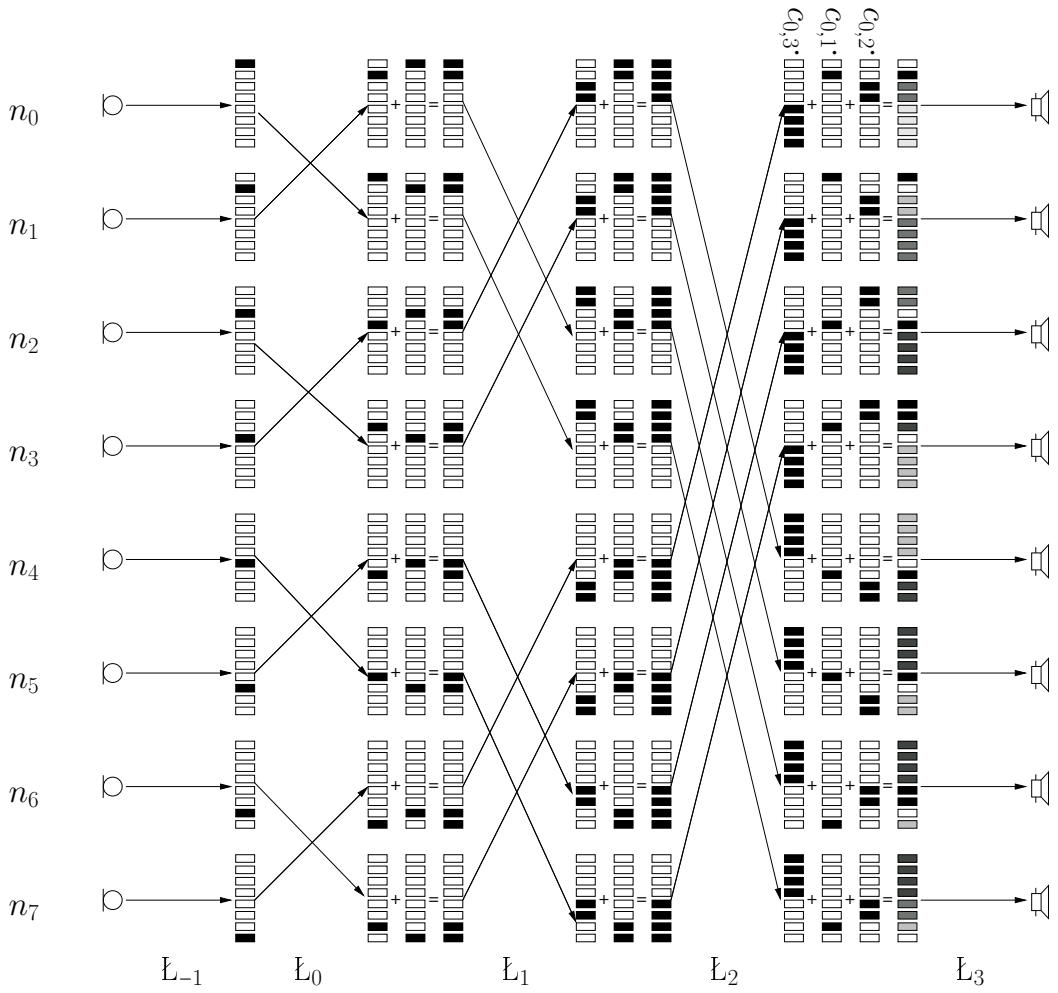


Figure 8.3.2: Hypercube aggregation scheme suitable for audio communication in AVEs

however, their exact volume and latency are of lesser importance.

To account for bandwidth constraints, the algorithm should minimize the number of levels, i. e. the degree of the network.

With avatars moving, peers dropping out and so forth, the topology needs to be modified continuously. Therefore, the algorithm should be stable in the sense that, e. g. a single person walking around in the virtual environment should have only local repercussions instead of affecting the whole network structure.

Obviously, peers that are neighbors in the network experience minimum latency and transcoding artifacts between each other. Particularly, peers that are neighbors on Ł$_0$ additionally have full individual channel control, i. e. highest possible granularity. Therefore, Algorithm 8.1, proposed below, clusters all participants with virtual locations referred to as *points* $p \in P$ standing in general position, i. e. in no special configuration, into a minimum diameter network graph with the property that participants standing closest together become neighbors on Ł$_0$. Figure 8.3.3 shows the clustering of six points according

Algorithm 8.1 *Clustering Method 1*

$T := \{\{p\}|p \in P\}$ *// Initialize working set to singleton clusters from points.*
While $|T| > 1$
// Cluster mutually closest clusters...
$D := \left\{\{u, v\} \in T^2 | u \neq v, \forall x \in T \backslash \{u, v\} \left(|u - v| < |u - x| \wedge |u - v| < |x - v|\right)\right\}$
$T := T \backslash \{x \in c | c \in D\}$ *// and remove from working set.*
$C := \{\{u\}|u \in T\}$ *// Form singleton clusters from remaining clusters.*
$T := D \cup C$ *// Recurse with combined result.*
end

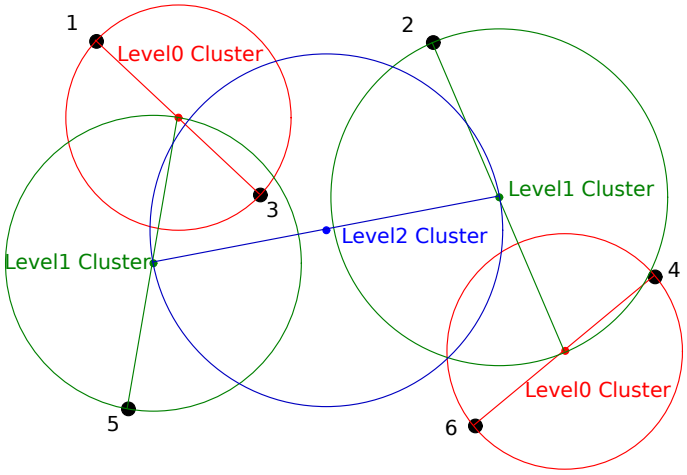


Figure 8.3.3: Clustering six points by Method 1

to Method 1. Mutually nearest points $\{1, 3\}$ and $\{4, 6\}$ are clustered first (red circles). Remaining points 5 and 2 are clustered into singletons (not shown). In the next loop, the mutually closest clusters are clustered again (green circles). This is repeated a final time, leaving a single resulting cluster, the *root cluster* (blue circle). As a result, point 1 could have the following neighbors (point IDs): 3 (on Ł$_0$), 5 (on Ł$_1$), 4 (on Ł$_2$). The disadvantage of Method 1 is that the high number of singleton clusters (e. g. $\{2\}$ and $\{5\}$) results in a high node degree. As a consequence several variations were devised:

Method 2 differs from Method 1 in that it keeps clustering on the same level as long as possible. In the example, it would cluster $\{2, 5\}$ even though they are not particularly close. This results in a lower degree, however, at the expense of scene accuracy. As a compromise, four additional methods refine Method 2 by allowing the clustering of remaining points only if additional constraints are met:

Method 2.1 requires the centers of the two clusters to be within audio range.

Method 2.2 requires at least one pair of points with the members from both clusters to be within audio range.

Method 2.3 requires the circular cluster perimeters to be within audio range.

Method 2.4 refines Method 2.1 by additionally requiring the two clusters' unilaterally closest points to be of approximately the same distance (e. g. up to a factor of 2) to both clusters.

8.4 Further Considerations for the Implementation

Architecture The proposed realization of scalable audio communication for MMVEs is a peer-to-peer network where each network node, called *peer*, communicates with a small set of other peers, called his *neighbors*, by periodically exchanging UDP packets containing sequences of partially mixed audio data. On the large scale, network nodes are arranged in an overlay graph consisting of—generally incomplete—hypercube networks *(cells)* connected by a, e. g. hexagonal, grid network. The internal architecture of each peer comprises an input layer, a delivery layer, an aggregation layer and an output layer, as detailed in Chapter 7.

Timing Considerations Several pitfalls must be avoided when implementing this seemingly simple architecture. Audio streaming is a real-time process. From the recording to the encoding, sending, receiving, decoding, aggregating to the playback, all operations need to complete within time bounds to avoid loss of streaming. On real-time operating systems, this is ensured by guaranteeing periodic CPU time slices to real-time processes. On non-real-time operating systems, other processes can block the CPU or obstruct I/O paths; the uninterrupted operation is therefore not warranted. Nevertheless, operation can be improved by a considerate implementation that mitigates real-time requirements as much as possible. On this account, all CPU-intensive tasks, such as audio mixing or loss concealment, are to be performed well before the send deadline where possible.

Sound I/O and Spliced Mode In a PC, the input (*recording*) and output (*playback*) of sound is performed by a dedicated hardware component, called the *sound card*. It consists mainly of a *digital-to-analog converter (DAC)* for playback and an *analog-to-digital converter (ADC)* for recording, each connected to a small buffer. Samples are transferred to main memory using direct memory access (DMA) operations.

As particularized in Subsection 6.2.3 et seq., Spliced Mode and Chained Mode synchronize all input (i. e. record) and output (i. e. playback) operations to the network, separately for each level. For this, independent phases are needed both for recording and playback. Independent I/O streams can be approximated by using a small sampling period for communication with the sound card and assembling differing sets of consecutive small periods into bigger packets handed to each accumulator. The minimum size of the sampling period is determined by the sound card hardware and the bus load of the system. If the bus is occupied for a longer time than the sampling period, loss of streaming will occur. If the application is notified at each DMA transfer, a small period will lead to a high CPU load, that will also increase the risk of buffer underruns.

The Advanced Linux Sound Architecture (ALSA) supports two modes of transferring data between the application and the ALSA system:

- Standard I/O transfers using `read()` and `write()`, possibly in conjunction with `select()` and `poll()`.
- Memory-mapped I/O, i. e. direct access to the ring buffer.

Using the latter method for the implementation has several advantages: Phase shifted local source ports/input channels may simply read different, mostly contiguous regions of the ring buffer. The same is true for local destination ports/output channels, with the additional advantage that slightly belated audio packets may still be written to the ring buffer, so that only a few samples are missed. Furthermore, the memory-mapped access mode allows for zero-copy communication, because ALSA does not need to copy the samples from application to another place in system memory [91].

8.5 Evaluation

In this section, scene accuracy and mouth-to-ear latency of the approach will be subjected to quantitative analysis.

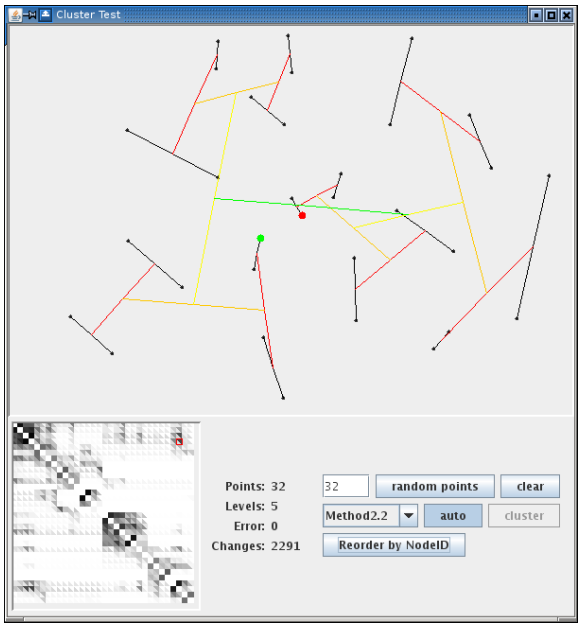


Figure 8.5.1: Evaluation tool for clustering/mapping algorithms

8.5.1 Mapping

To evaluate the performance of the mapping algorithms presented in Section 8.3, a JAVA Swing based evaluator was developed. Its graphical user interface is shown in Figure 8.5.1. In the upper portion, configurations of virtual positions can be entered. Using the controls at the lower right, these configurations can be clustered according to any of the six methods regarded above. Points being put in one $Ł_0$ cluster are shown as connected by a black line; lines of colors red, orange, yellow, and green illustrate clustering of clusters on levels 1, 2, 3, and 4, respectively. The resulting errors (artifacts) are visualized in the panel in the bottom left: For every speaker/listener pair, the coefficients are visualized, whereat black represents maximum loudness and white, inaudibility. The diagonal of white squares corresponds to the fact that no participant can hear himself. The upper right half (triangle)

of each square represents the nominal coefficient, and the lower left, the actual coefficient as it would result from the current clustering. This gauge helps to quickly spot problems associated with a particular clustering. For quantitative evaluation two experiments can be initiated through the command line interface:

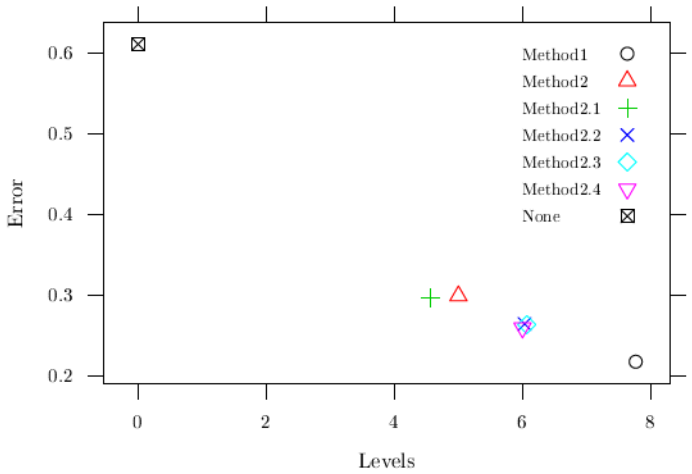


Figure 8.5.2: Scene accuracy and degree subject to clustering method

In the first experiment, random configurations of 32 points are generated and clustered according to the six methods. The total error and maximum node degree are recorded. The results are shown in Figure 8.5.2. Method 1 is shown to yield the most accurate audio reproduction, however, it leads to a high node degree of almost 8 on average. Method 2 manages with a node degree of 5, which is optimal for a contiguous network 32 nodes. However, the error is about 50% higher, too. Method 2.1 performs slightly better, both with regard to error and degree. Methods 2.2, 2.3, and 2.4 form a compromise about half-way in between Method 2.1 and Method 1.

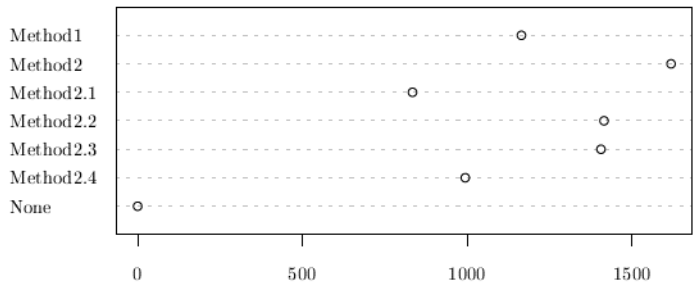


Figure 8.5.3: Structural changes incurred by avatar crossing the scene

In the second experiment, one point is moved through the configurations of points, and the resulting changes, i. e. adding and removal of neighbors, are recorded. The results are presented in Figure 8.5.3. Method 2.1 is shown to be the most stable one, whereas Method 2, which optimizes levels most aggressively, unsurprisingly comes last.

8.5.2 Mouth-to-Ear Latency

Mouth-to-ear latency of the approach was determined using the prototypical implementation from Chapter 7 and compared to the analytical expectations and upper bounds from Chapter 6. *Mouth-to-ear latency (M2E latency, MEL)* denotes the time from the recording of samples at the source to their playback at their destination. Obviously, MEL performance strongly depends on the system's underlying network environment. To reproduce realistic Internet characteristics in the evaluation as closely as possible, the application was run on top of the OMNeT++ simulator in conjunction with the INET Framework and the DELFOI delay model presented in Chapter 4, that accounts for network locality, inter-region specific jitter, individual Internet access link bandwidths and queueing. Parametrization of the delay model was performed from Internet measurements as described ibidem. From the data set random hosts located in Germany were drawn. Non-realtime operating system behavior was modeled by Erlang ($k = 2$, $mean = 2$ ms) distributed response times to timer and sound card interrupts. The application's timing discipline was set to Chained Mode (Subsection 6.2.4). Assuming a 16 kbps bitrate codec for good voice quality in conjunction with a packet length of 50 ms, i. e. 20 packets à 100 bytes UDP payload plus headers per second, yields roughly 25 kbps bandwidth required to support one connection. As the record buffer length a conservative value of 25 ms, i. e. $k = 2$ was chosen. To study the effect of network size on MEL, the number of nodes N was varied from 2 to 256. The minimum RTT was determined for each neighbor pair and its PDF computed. From this I calculated, for complete hypercube allocations, the upper latency bound (symbol $\triangle$), i. e. the sum of T, (6.1.1), and (6.2.9); and the expected value (symbol $\circ$), i. e. the sum of T, (6.1.2), and (6.2.8). For each source/destination pair, i. e. not only direct neighbors, I measured MEL, took the median of the last 200 measurements to eliminate initialization effects, and computed mean (symbol $\times$) and the percentiles P25, P75, and P99 (symbol $\bullet$) over all pairs. For each parametrization of N, all results were averaged over 50 simulation runs with varying random hypercube graphs produced from randomly drawn hosts. The results are plotted in Figure 8.6.1.

Measured MEL mean is found to be slightly larger than its expectation, likely because of queueing delays and imperfect hypercubes due to bandwidth constraints. First and third quartile lie close by. The more delicate 99^{th} percentile stays still below the analytical bound. For one network with 64 nodes, I examined the situation more closely. Figure 8.6.2 shows the upstream bandwidths available. Note that two nodes do not possess sufficient bandwidth (vertical line) to join a network of degree six. To support these nodes anyhow, the network topology had to be transformed from a perfect hypercube of dimension six to an incomplete (sparse) hypercube of higher dimension.

Figure 8.6.3 shows the resulting MEL distribution for this network in detail. Though the mean exceeds the expectation, still 88% end-to-end latencies are below 500 ms.

8.6 Conclusion

I have presented an approach towards scalable audio communications in MMVEs using latency optimized hypercube gossiping, complemented by analytical latency bounds, architecture and implementation considerations. For domestic connections, results based on Internet measurements from the year 2010 exhibit, for networks of 64 nodes, latencies below 500 ms for 50 of the conversation partners. Among the mapping algorithms, Method 2.1 yields the best overall performance. However, the number of structural changes due to one moving avatar is still considerably high. Future work should consider excluding

avatars in full motion from the regular scene mapping.

Obviously, these quantitative results can only be preliminary. Whether current latency, accuracy and reconfiguration values are acceptable will have to be determined by user studies, whose outcomes, in turn, will likely depend on the specific application and associated user expectations.

Ultimately, when complemented by a stable mapping and the inevitable increase of consumer Internet bandwidth, exponentially amplified by the hypercube, the approach brings efficient and satisfying massively multiuser auditory virtual environments into close reach.

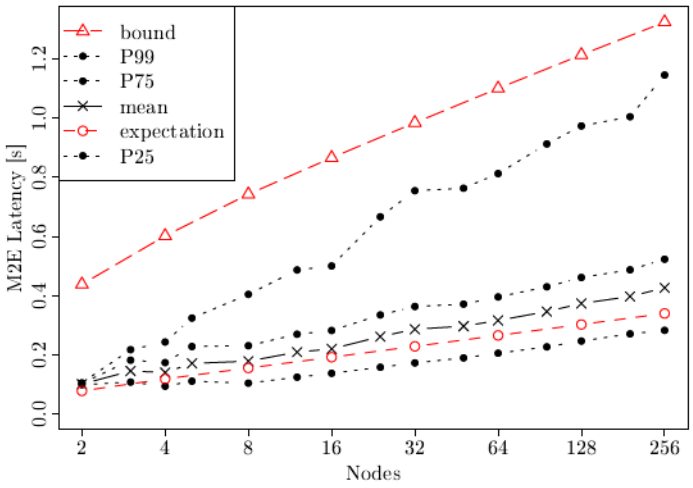


Figure 8.6.1: Mouth-to-ear latency for different network sizes

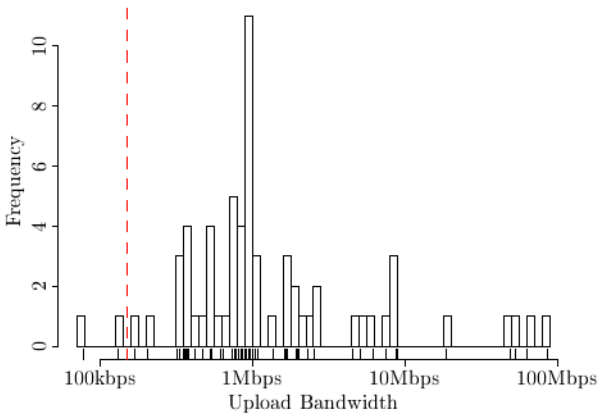


Figure 8.6.2: Upstream bandwidth for a specific network configuration of 64 nodes

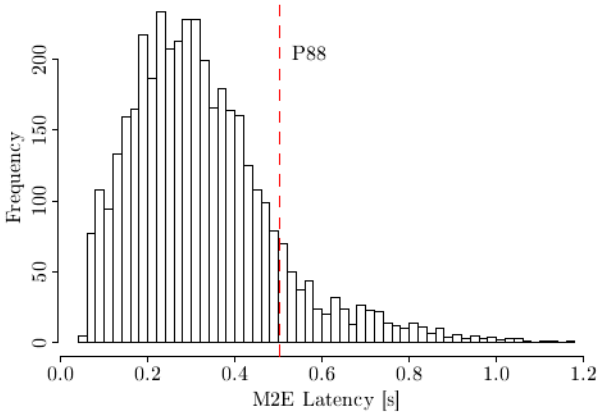


Figure 8.6.3: Mouth-to-ear latency distribution of the 64 node network

Chapter 9

Application 2: Decentralized Data Fusion for Object Tracking

Contents

Decentralized data fusion (*DDF*) is an instance of the general distributed inference problem, in which there is a single common state of interest—often a geographical area with an initially unknown number of physical objects (*targets*), their locations and velocities—that is observed by a number of distributed nodes, each equipped with one or more sensors, and connected by some (heterogeneous) network [106]. Each node periodically obtains observations from its sensors by which it estimates the state of interest. All nodes' estimates are merged (*fused*), along with previous estimates, to form a condensed estimate at each node that is more confident than one that any node could produce alone. In recent years, the decentralization of data fusion techniques to combine data from multiple sensors has received increasing attention. It coincides with the emergence of a new driving paradigm of *Network Centrics*, e. g. Architectures or Operations (NCO) that scatter responsibility across the network. In 1994, guiding principles were postulated [57], amounting to the abolishment of central components such as single fusion centers, central nodes, common communication facilities, or total knowledge of network topology. The rationale behind it is to ensure scalability by absence of central bottlenecks, as well as modularity and flexibility

by obviating the need for knowledge of the network. Apparently however, adherence to these principles is necessary but not sufficient to ensure scalability.

Scaling DDF networks in a heterogeneous environment presents several challenges: Bandwidth limits the number of neighbors to which each node may send state estimates. A fully connected topology, for instance, is therefore only possible for a small number of nodes. Providentially, data fusion is an operation that can reduce data volume. By performing the aggregation on the network, information can be condensed along the way. However, each extra hop introduces additional latency that is comprised not only of network delays but also of sojourn times of the information at intermediate nodes. This vitiates the value of state estimates; particularly estimates of targets that behave in a non-linear way may decay quickly. As a consequence, latency must be kept to a minimum. In [30], the necessity to spread information "as fast and as efficient as possible" and the need for analytical performance models and metrics to predict latency and reliability is emphasized.

Another challenge generally encountered in DDF networks springs from the necessity to eliminate common past information resulting from the fusion of estimates that were descended from the same data. To avoid double-counting, this correlated information from past fusion events must be identified and removed—a task which requires the availability of the original data sets. Although the theoretic fundamentals have been well documented and studied for about twenty years now, the removal of common information while minimizing the amount of data exchanged still remains one of the main difficulties in distributed information fusion [29].

Numerous methods for DDF have been proposed, handling correlated data in diverse ways. Yet all in all, current approaches compensate this so-called *data incest* by approximations, have scalability issues with regard to latency or bandwidth, are fragile regarding single node failures, or resort to central components.

In this chapter, I present a periodic peer-to-peer hypercube gossiping based approach to decentralized data fusion. Its basic hierarchical structure permits efficient aggregation and dissemination of information with logarithmic time and bandwidth requirements while ruling out the possibility of double counting. In contrast to 1-trees, such as star and chain topologies, the dense structure of a hypercube sustains connectivity in the event of single node failures. The responsibilities of all nodes are fully determined by their position in the hypercube, thus obviating the need to negotiate communication schemes or carry information pedigree logs. As default timing discipline, I propose Sync Mode, as per Subsection 6.2.2. For operation in severely bandwidth restricted environments, I present two aggregation-in-time schemes, based on Spliced Mode (Subsection 6.2.3) and Chained Mode (Subsection 6.2.4) to further decrease bandwidth requirements while scheduling network transmissions so as to keep latency at a minimum.

The rest of this chapter is structured as follows: Section 9.1 summarizes the theoretical background on sensor fusion, finite set statistics, the Bayes recursion and its first moment reductions, as well as the problem of double-counting. Section 9.2 shows how the problem is addressed in related work. Starting from the declaration of assumptions and the re-statement of the DDF problem, Section 9.3 presents the proposed periodic peer-to-peer hypercube gossiping based approach. Section 9.4 points out practical considerations regarding the accommodation to severely limited bandwidths while keeping latency minimal. Section 9.5 concludes this chapter.

9.1 Distributed Data Fusion

9.1.1 Finite Set Statistics (FISST)

With the introduction of the multi-target calculus of finite set statistics (FISST) in 1994 [103], new conceptual and computational methods to provide a powerful multi-sensor/multi-target object tracking have become available. Since then, these methods have received significant attention in the context of both military and non-military applications. FISST is a mathematical framework based on Bayesian statistical modeling for dealing with multi-sensor, multi-target and multi-evidence data fusion problems. In a nutshell, it bundles multiple sensors into a single meta-sensor and multiple targets into a meta-target while retaining their characteristics [103]. Using FISST, the state of a multi-target system is described by a finite set of the form $X = \{x_1, \ldots, x_M\}$, where M is the number of targets, and $x_1, \ldots, x_M$ are the state vectors of individual targets. The state is assumed to be a Markov process, whereat $f_{k|k-1}(X_k|X_{k-1})$ is the probability density that state X_{k-1} will transition into X_k at time k. The Markov process is observed by sensors modeled by a *likelihood function* $g_k\left(Z^k|X_k\right)$, purporting the probability density of receiving the observation Z^k given a state X_k at time k. The probability density of the state X_k at time k, given all observations $Z^{1:k} = (Z^1, \ldots, Z^k)$ up to time k, is denoted by $p_k\left(X_k|Z^{1:k}\right)$. It is derived from the propagation of a (multi-target) posterior in time, known as the *Bayes recursion* [148]:

$$p_{k|k-1}\left(X_k|Z^{1:k-1}\right) = \int f_{k|k-1}(X_k|x)\, p_{k-1}\left(x|Z^{1:k-1}\right)dx \tag{9.1.1}$$

$$p_k\left(X_k|Z^{1:k}\right) = \frac{g_k\left(Z^k|X_k\right) p_{k|k-1}\left(X_k|Z^{1:k-1}\right)}{\int g_k\left(Z^k|W\right) p_{k|k-1}\left(W|Z^{1:k-1}\right) dW} \tag{9.1.2}$$

In multi-sensor tracking, the state of interest is observed by multiple sensors whose estimates all contribute to the total estimate. Two estimates $p(X_k|Z_i)$ and $p(X_k|Z_j)$ of a common state X_k at time k but conditioned on different observations Z_i and Z_j can be fused using the *fusion rule* [32]

$$p_k\left(X_k|Z_i \cup Z_j\right) = C^{-1} \frac{p_k\left(X_k|Z_i\right) p_k\left(X_k|Z_j\right)}{p_k\left(X_k|Z_i \cap Z_j\right)} \tag{9.1.3}$$

where C is a normalizing constant.

9.1.2 First Moments

That being said, due to immense computational demands, full Bayesian multi-target filtering is generally not applicable. For this reason, usually only a multi-target first-order moment statistic, called *probability hypothesis density (PHD)* or *intensity*, of the multi-target posterior is considered [102]. Accordingly, the *PHD recursion* propagates the posterior intensity $D^{k|k}$ of the random finite set of targets in time. The expected number of targets M can be obtained by integrating $D^{k|k}$ over the state space. The actual targets are then localized by extracting the M highest peaks [49]. Problematically, the thus computed number of targets exhibits a high variance. For this reason, the PHD recursion was generalized to the *cardinalized PHD (CPHD) recursion*, which jointly propagates the posterior of the posterior cardinality distribution [147] and thereby provides more stable estimates as to the number of targets M. The (C)PHD is commonly represented by particles [146] or more efficient Gaussian Mixtures [148]. Some closed form solutions were presented in [104].

9.1.3 Double Counting

One particular problem in decentralized data fusion arises from *data incest*, i. e. the fusion of estimates that were descended from the same data and are therefore not conditionally independent anymore. To avoid *double-counting* this common information, it needs to be canceled out by regarding it in the denominator of (9.1.3). However, the computation of $p(X_k|Z_i \cap Z_j)$ is generally intractable [144]; hence, great effort has been put into its approximation by numerous approaches.

9.2 Related Work

In [57], the problem of double counting is first considered for different topologies (fully connected, tree connected and networks with multiple paths), and the concept of *Channel Filters (CF)* for the removal of common information in tree connected topologies is developed. Installed into each communication link, a Channel Filter maintains the common past information between the two nodes on each side of the link on the basis of information communicated through the channel. More recently, [117] presents an approach for performing the associated DDF operation of division by way of importance sampling. Still and all, for topologies that have multiple, redundant paths between nodes, the Channel Filter approach will not produce consistent information state estimates because each filter only aggregates information that flows through it. Particularly, it is not generally possible to find a consistent centralized estimate for each node in an arbitrary network topology based on local information only [57]. The *Information Graph (IG)* technique, presented in [32], is a symbolic representation of the collection, propagation, and fusion of data and can be used to identify common information in distributed estimation systems. However, considering all relevant common priors and removing the common information at the nodes requires carrying steadily growing pedigree logs that include communication and fusion events as well as past fusion data, which is not applicable in the face of limited bandwidth [29].

A different path is pursued by the geometrically inspired *Covariance Intersection (CI)* [74] data fusion algorithm that takes a convex combination of the means and covariance of the sources in information space. CI has the advantage that it does not require specific information about the correlation but is limited to Gaussian distributions. *Exponential Mixture Densities (EMD)* as generalization of CI for general probability distributions as well as multi-object posteriors were independently proposed in [101] and derived in [67]. A mathematically tractable representation and realization for multi-object tracking was first presented in [34].

Another class of approaches counters the problem by use of special topologies. [137] proposes two methods based on independent cliques whereas "each clique receives a partial estimate from its preceding clique, and updates the partial estimate with local observations made within the clique. Then, it forwards the updated estimate to the next clique". This pipelined distribution is scalable with regard to bandwidth but leads to a high latency. The authors' second algorithm entails forwarding the estimates to a fusion center to obtain the final output.

The approach that is closest related to the one proposed is [143]. Instead of compensating the effects of data incest, the approach avoids it in the first place by keeping distinct data-tagging sets and exploiting the tree nature of the proposed 2-tree topology for optimization. The mesh topology of the 2-tree network appears to be a particular fit for wireless (ad-hoc) sensor networks. However, provided that all nodes can reach each other and observe a common region of interest, more latency efficient topologies are possible.

9.3 DDF on Hypercubes

The background given in Section 9.1 being understood, we can disregard the actual representation of posteriors as well as the cardinality, type, transition, and distribution of targets and focus on the dissemination of that information instead.

9.3.1 Assumptions

We assume that a set Ω of $N = |\Omega|$ distributed nodes are observing one common state of interest with sensors emitting observations at equal time intervals. Each node has unrestricted reachability, i. e. it can reach every other node, features full-duplex communication, i. e. can send and receive at the same time, but has only limited bandwidth, i. e. can only communicate with a limited number of peers per time interval and direction. Particularly, we assume both measurement errors and process noise of all nodes to be uncorrelated. The former depends on faithful modeling of all sensors and their likelihoods whereas the latter requires a true target motion model; neither of which may be generally available.

9.3.2 The DDF Problem

Given an initial probability density $p\left(X_0|Z_\Omega^0\right)$ and observations $Z_\Omega^{1:k}$ becoming available step-by-step at the sensors, the ultimate goal is to have, at all times k and at every node n, an estimate conditioned on as many observations as possible.

Based on (9.1.1), (9.1.2), (9.1.3) we define three abstract operations:

1. Generation G of a new posterior from one observation:

$$p\left(X_k|Z_{\{i\}}^k\right) = \mathrm{G}\left(Z_i^k\right) := \frac{g_k\left(Z_i^k|X_k\right)p\left(X_k\right)}{p\left(Z_i^k\right)}, \tag{9.3.1}$$

2. Transition T of one posterior in time:

$$p\left(X_k|Z\right) = \mathrm{T}\left(X_{k-1}|Z\right) := p_{k|k-1}\left(X_{k-1}|Z\right) = \int f_{k|k-1}\left(X_{k-1}|x\right)p\left(x|Z\right)dx, \tag{9.3.2}$$

3. Fusion F of two conditionally independent posteriors of the same time, conditioned on observation sets $I, J \subset \Omega$:

$$\begin{aligned} p\left(X_k|Z_{I\cup J}\right) &= \mathrm{F}\left(X_k|Z_I, X_k|Z_J\right) := p\left(X_k|Z_I \bigcup Z_J\right) \\ &= C_1^{-1}\frac{p\left(X_k|Z_I\right)p\left(X_k|Z_J\right)}{p\left(X_k|Z_I \cap Z_J\right)} = C_2^{-1}p\left(X_k|Z_I\right)p\left(X_k|Z_J\right). \end{aligned} \tag{9.3.3}$$

With regard to the sequence of operations $\{\mathrm{G}, \mathrm{T}, \mathrm{F}\}$, several solutions are possible to reach the final result $p\left(X_k|Z_\Omega^{(k)}\right)$.

9.3.3 Hypercube Gossiping

Information from all sensors needs to be disseminated to all nodes. The DDF problem can thus be regarded as an instantiation of the classic *gossip problem,* defined in Subsection 2.1.3. Assuming limited access bandwidth, we regard the *full-duplex* or two-way

communication model where each node may both send and receive data to/from one node at a time. In *stream gossiping*, the exchange is performed not once but continuously, periodically. In the generic periodic peer-to-peer gossiping approach proposed in Chapter 3 of this thesis, hypercubes were chosen for their simplicity and gossip efficiency—properties that also render them particularly well suited for the DDF problem at hand. We adopt the communication and aggregation schemes from Definition 3.4.12 and 3.4.13. The information sent by node n on level ℓ consists of a state estimate $\hat{X}_k(n,\ell)$. The amount of information $I_{\mathrm{L}_\ell}(n)$ increases with ℓ as the estimate is fused with estimates from other nodes.

Dissemination Scheme

In periodic intervals, e. g. after every observation, each node sends estimates to its neighbors. Specifically, node n sends to each neighbor $\mathrm{nb}(n,\ell)$ the fusion of the posteriors received on lower levels $[0..\ell-1]$ and the posterior $p\left(X_k|Z_n^k\right)$ generated from the latest observation Z_n^k. The resulting communication scheme is depicted in Figure 9.3.1 for a network of eight nodes. Each dimension of the hypercube corresponds to a different cardinality of accounted observations Z. All communicated estimates are conditioned solely on the most recent observations of all contained sources, i. e. they do not include estimates from prior observations.

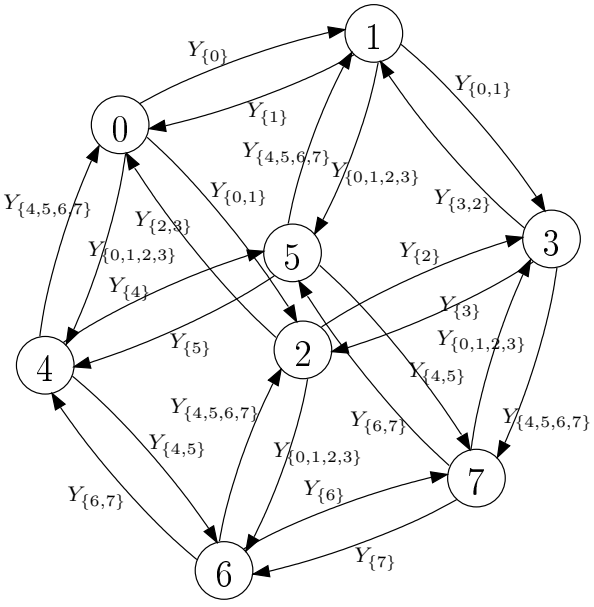


Figure 9.3.1: Posterior dissemination among nodes in complete H_3 network. For the sake of readability, $p(X|Z_J)$ is denoted Y_J.

Fusion Scheme

In accordance with Definition 3.4.10, the estimate $\hat{X}_k(n,\ell)$ to be sent from node n to its neighbor on level ℓ is given by the recursive function

$$\hat{X}_k(n,\ell) := \begin{cases} \mathrm{G}\left(Z_n^k\right) & \text{when } \ell = 0 \\ \mathrm{F}\left(\mathrm{T}^{k-h}\left(\hat{X}_h\left(\mathrm{nb}(n,\ell-1),\ell-1\right)\right),\ \hat{X}_k(n,\ell-1)\right) & \text{otherwise} \end{cases}$$

where the functional power T^n denotes n-fold iteration of T.

In Figure 9.3.2, the corresponding fusion architecture is shown by example of node 0. At time k, the sensor emits a new observation Z_0^k from which a new posterior $p\left(X_k|Z_0^k\right)$ is generated (G) and sent to node 1. Received posteriors from nodes 1, 2, and 4 are transitioned (T) to time k and fused (F) before being relayed via the next higher level to nodes 2 and 4.

In addition to these $\log_2(N)$ current estimates being sent to neighbors, at each node n, another estimate $\breve{X}_k(n)$ is fused from all collected information, this time also including past observations, to form the total posterior. In contrast to the audio communication application regarded in Chapter 8, the DDF application requires a fusion result that includes the data from each node's own sensor as defined in (3.4.6). For DDF, this fusion is depicted by the filled fusion node and defined as

$$\breve{X}_k(n) = \mathrm{F}\left(\mathrm{T}\left(\breve{X}_{k-1}(n)\right),\ \hat{X}_k(n,L-1)\right). \tag{9.3.4}$$

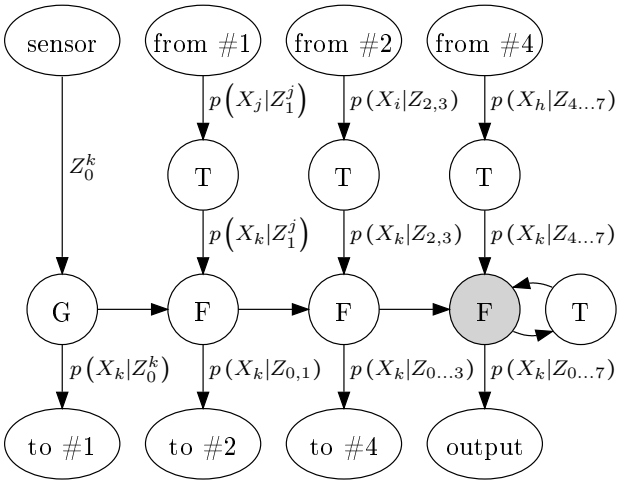


Figure 9.3.2: Fusion scheme, node 0 of 8

9.3.4 Degeneration

For incomplete allocations, we adopt the scheme from Subsection 3.3.2 and Subsection 3.4.2.

Node Failures

Robustness requirements of DDF demand that systems remain functional in the event of single node failures. For singly-connected or 1-tree networks, such as star and chain topologies, the failure of any non-leaf node will separate the network in two components [143]. In denser networks without redundant paths, like the proposed hypercube topology, node failures will result in the non-appearance of estimate updates at other nodes but not break the network. As shown in Theorem 3.4.11, an instantaneous node failure in a previously complete L-level network with 2^L nodes will, until corrective action is taken, result for 2^i nodes in the loss of 2^{L-i-1} sources with $i \in [0, L-1]$. This will reduce the estimation performance of some nodes until reconfiguration but, assuming $L \geq 2$, no node will become isolated.

Node Arrival

Joining the network must not require knowledge of the network as a whole. The position at which the new node is to join the network can be determined from the latest estimates received by any node: Upon generation, estimates are tagged with the respective source IDs whereas fusion results are tagged with the set union of IDs of fused sources. The join address is then determined as the minimum node ID whose source is not present in the estimates. The neighbors of that position can be reached from any node in $\log N$ steps.

Nodes joining the network have no knowledge of past observations. For most practical applications the estimate fused from recent observations should quickly converge to the total estimate based on all past observations. Accordingly, a joined node should have reasonable confidence in little time. To further speed up convergence, a new node can be given the total estimate of any node. In this case, care must be taken not to add estimates already contained in this total. To this end, upon generation, new estimates are tagged with sensor-unique consecutive sequence numbers. Only estimates that have higher sequence numbers for all mutual sources may be fused into a (time-aggregated) estimate. Note that this simple tagging does not incur the cost of pedigree logs, which grow in time.

9.4 Optimization

As shown in Chapter 6, wait latency is determined by the timing of all nodes' transmissions.

Sync Mode

A basic, natural choice is to perform the transitioning of estimates upon their reception and the generation, fusion, and transmission upon the emission of a new observation by the sensor. This will at each node result in expected wait delays of $\delta = 1/2\,T$, and for every information source in an expected mean latency of all sources, as experienced by one node, of

$$E[\Delta_{\text{Sync}}] = \frac{\sum_{h=1}^{L} \binom{L}{h} h}{N} \frac{T}{2} = \frac{LT}{4}. \tag{9.4.1}$$

Note the simpler form compared to (6.2.3) due to the inclusion of the node's own sensor data.

9.4.1 Coping with Scarce Bandwidth

The bandwidth required for each direction grows in the order of $\log N$. Below, we will point out considerations for coping with bandwidths that are insufficient to implement the basic scheme laid out in Section 9.3.

Single Nodes with Reduced Bandwidth

Single nodes with insufficient bandwidth can be accommodated by relocating them up in ID space, into the sparser regions of the (incomplete) hypercube, as described in Subsection 3.3.4. Generally insufficient bandwidth calls for a different approach.

Insufficient Bandwidth in General

All posteriors that are sent at Level 0 are based on just one observation each. In case of observations with high clutter rates, the transmission of these noisy posteriors may use up a considerable portion of the available bandwidth. Instead of sending posteriors after every observation, estimates from s successive observations may be fused together before being sent. By using this aggregation in time, bandwidth is reduced not only by sending packets less often but also because uncorrelated clutter may be removed from the (e. g. Gaussian mixture) approximation to be transmitted, thereby reducing data volume. The resulting extended period shall be denoted $T' = s\,T_{\text{Slot}}$, where T_{Slot} denotes the interval between observation emissions, formerly used for T.

Spliced Mode

When using aggregation in time in conjunction with Sync Mode (6.2.2), wait latency as of (9.4.1) is multiplied by s. By updating the final estimate every T_{Slot} instead of T', wait delay at the destination stage is retained and total latency reduced to

$$\begin{aligned} E\left[\Delta_{\text{Spliced}}\right] &= \frac{\sum_{h=1}^{L}\binom{L}{h}h + \left(2^L-1\right)\left(\frac{1}{s}-1\right)}{N}\frac{T'}{2} \\ &= \left(\frac{L}{2} - \frac{2^L-1}{2^L}\left(1-\frac{1}{s}\right)\right)\frac{T'}{2}. \end{aligned} \tag{9.4.2}$$

Chained Mode

The sequence of transmissions along an overlay path of maximum length is called a *chain*. To further reduce wait latency along these chains as much as possible, each $Ł_{\ell+1}$ estimate is sent upon the reception of the next observation following the reception of the respective $Ł_\ell$ estimate. By using differently grouped sets of consecutive observations for the estimates to be sent on different levels, wait delay within chains remains $1/2\,T_{\text{Slot}}$ and only chain switching delays will increase to $1/2\,s\,T_{\text{Slot}}$, resulting in a total of

$$\begin{aligned} E\left[\Delta_{\text{Chained}}\right] &= \frac{\sum_{h=1}^{L}\binom{L}{h}h\frac{1}{s} + \mathrm{g}\left(L\right)\left(1-\frac{1}{s}\right)}{N}\frac{T'}{2} \\ &= \left(\frac{L}{2s} + \frac{\mathrm{g}\left(L\right)}{2^L}\left(1-\frac{1}{s}\right)\right)\frac{T'}{2} \end{aligned} \tag{9.4.3}$$

with $\mathrm{g}()$ as per (6.2.7) counting among all data paths the sub-paths that contain one or more skipped links in between taken links.

In Figure 9.4.1, expected wait latency $E\left[\Delta\right]$, relative to T_{Slot}, is plotted for different network sizes, aggregation factors s, and timing modes Sync, Spliced, and Chained. Visibly, latency is roughly proportional to both the number of levels $L = \log_2 N$ and the aggregation factor s, with Chained Mode resulting in a considerably shallower increase with both parameters.

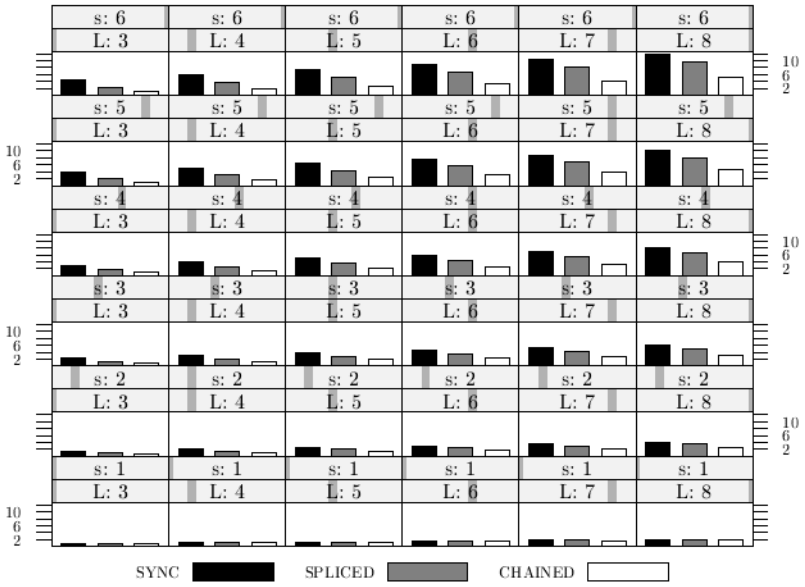


Figure 9.4.1: Relative Mean Wait Latency as a function of Network Size L, Aggregation Factor s, and Timing Mode

9.5 Summary

In this chapter, I have presented a scalable approach to distributed data fusion based on periodic peer-to-peer hypercube gossiping. Being a minimal gossip graph, the hypercube topology exhibits only logarithmic bandwidth requirements and latency. Its simple hierarchical structure allows for efficient network reconfiguration and avoidance of double counting with local knowledge only and no pedigree logs. Just like Channel Filters, the proposed approach requires all nodes' estimation errors to be independent. In addition, I have presented strategies for coping with severely limited bandwidth while keeping latency at a minimum.

Currently, the presented approach is limited to binary hypercubes, which provide optimal scalability for meager bandwidths. Generalization to arbitrary bases allows to make full use of abundant bandwidth to further reduce latency.

Chapter 10

Conclusion

In this thesis I have considered the combined dissemination and fusion of streaming information from multiple sources. At this, every node needs, at all times, an aggregate fused from each node's as-current-as-possible information.

Although this task could be viewed as a specialization of the classic gossip problem, at which each node's single piece of information is communicated to all other nodes, it may well be considered a fundamental problem of its own. Neither is it characterized by the gossiping of one fusion result, nor by the fusion of several gossiping results. Moreover, the fusion-gossiping of streaming information is unarguably entirely different from the repeated execution of the one-shot gossiping scheme. Indeed, the regarded problem has numerous applications in diverse areas. Treating these applications to a composition of solutions for the conventional problems would misconstrue the nature of the problem and lead to suboptimal results, just as the gossip problem would generally be poorly solved by accumulation followed by broadcasting.

As frequently the case with live streaming applications, the applications regarded by this thesis demand up-to-date information with as little latency as possible. In most real-world scenarios bandwidth is a scarce resource, usually being regarded as a given. This refines the problem into finding a strategy that makes most efficient use of bandwidth with regard to network size and latency.

This thesis contributes towards an efficient solution with predictable performance, specifically, a system for the decentralized bandwidth-efficient low-latency fusion-dissemination of high-frequency periodic measurements in heterogeneous environments.

To this end, the first contribution is a peer-to-peer network architecture based on the versatile and efficient hypercube topology. The network is fully symmetric in its complete allocation yet not limited to certain sizes. It is robust with regard to single node failures and flexibly accommodates a minority of bandwidth impaired nodes. On this basis, gossiping communication and aggregation schemes are provided, facilitating gossiping and aggregation in $\log N$ communication rounds.

A separate area of contributions deals with the analytical quantification and reduction of latency, which depends to a considerable portion on transmission timing and, as such, is under direct control by the application. Four local timing disciplines and a globally optimizing one have been investigated, reducing wait latency in the portrayed scenarios by up to eighty percent. Analytical bounds and expectations for wait latency and resulting distribution of total traversal time have been developed and were shown to yield accurate results.

Supplemental contributions include an underlay network model parametrized from large-scale measurements and a real-time simulation and visualization tool for periodic communication. The former efficiently provides location- and load-dependent latency, jitter and packet loss samples for connections between arbitrary hosts world-wide and is hence particularly suited for the simulation and evaluation of peer-to-peer networks. The latter uses steady state analysis to emulate network timing and latencies of P2P networks.

Above contributions were completed by two exemplary applications, one for the provision of audio communications to massively multiuser virtual environments, and the other for distributed data fusion for live object tracking.

Bibliography

[1] Speedtest flow and methodology. `http://wiki.ookla.com/test_flow`, December 2011. Retrieved 2012-03-15.

[2] Bo Allen. Pseudo-random vs. true random - a simple visual example. http://www.boallen.com/random-numbers.html, April 2008. Retrieved 2012-02-13.

[3] D.P. Anderson. Boinc: a system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, November 2004.

[4] David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry, Springer New York*, 8(1):295–313, December 1992. 0179-5376 (Print) 1432-0444 (Online).

[5] P. Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, March 1964.

[6] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The new jersey data reduction report. *IEEE Data Engineering Bulletin*, 20:3–45, 1997. http://www.odysci.com/article/1010112984386637.

[7] Cagatay Basdogan and Ayam A. Srinivasan. *Handbook of Virtual Environments*, chapter Haptic rendering in virtual environments, pages 117–134. Lawrence Erlbaum Inc, 2002.

[8] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of the 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007*, pages 79–84, Anchorage, AK, USA, May 2007.

[9] Rene Beier and Jop Sibeyn. A powerful heuristic for telephone gossiping. In *Proceedings of the 7th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2000.

[10] Gordon Bell and Jim Gray. What's next in high-performance computing? *Commun. ACM*, 45:91–95, February 2002.

[11] Michel Berkelaar, Jeroen Dirks, Kjell Eikland, Peter Notebaert, and Juergen Ebert. lp_solve v5.5.2.0: A free mixed integer linear programming (MILP) solver. http://lpsolve.sourceforge.net/5.5/, August 2010.

[12] L.N. Bhuyan and D.P. Agrawal. Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on Computers*, C-33(4):323–333, April 1984.

[13] Danny Bickson and Dahlia Malkhi. The Julia content distribution network. In *Proceedings of the 2nd Conference on Real, Large Distributed Systems*, volume 2 of *WORLDS'05*, pages 37–41, Berkeley, CA, USA, 2005. USENIX Association.

[14] Danny Bickson, Dahlia Malkhi, and David Rabinowitz. Efficient large scale content distribution. In *WDAS '04: Proceedings of the 6th Workshop on Distributed Data and Structures*, Lausanne, Switzerland, July 2004.

[15] Jens Blauert. *Spatial Hearing: The Psychophysics of Human Sound Localisation.* MIT Press, Cambridge, MA, 2nd revised edition, 1996.

[16] Jens Blauert, editor. *Communication Acoustics.* Signals and Communication Technology. Springer-Verlag, 2005.

[17] Alfred Boals, Ajay Gupta, Jahangir Hashmi, and Naveed Sherwani. Compact hypercubes: Properties and recognition. In Frank Dehne, Frantisek Fiala, and Waldemar Koczkodaj, editors, *Advances in Computing and Information (ICCI '91)*, volume 497 of *Lecture Notes in Computer Science*, pages 395–402. Springer Berlin / Heidelberg, 1991. 10.1007/3-540-54029-6_187.

[18] Jean-Chrysostome Bolot. Characterizing end-to-end packet delay and loss in the Internet, 1993.

[19] Jean-Chrysostome Bolot and Sacha Fosse-Parisis. Adding voice to a distributed game on the internet. In *INFOCOM '98: Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 480–487, 1998.

[20] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, L. Haas, K. Kim, and N. Tatbul. Federated stream processing support for real-time business intelligence applications. In *BIRTE '09: Proceedings of the VLDB '09 Workshop on Enabling Real-Time for Business Intelligence.* Springer, 2009.

[21] P. Boustead and F. Safaei. Comparison of delivery architectures for immersive audio in crowded networked games. In *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 22–27. ACM, 2004.

[22] Raymond T. Boute. The euclidean definition of the functions div and mod. *ACM Trans. Program. Lang. Syst.*, 14(2):127–144, 1992.

[23] J. Allen Brack and Frank Pearce. The universe of World of Warcraft (keynote address). In *GDC '09: Game Developers Conference*, Austin, Texas, USA, September 2009. United Business Media LLC, NY.

[24] Fernanda Brandi and Eckehard Steinbach. Low-complexity error-resilient data reduction approach for networked haptic sessions. In *HAVE '11: Proceedings of thee IEEE International Symposium on Haptic Audio-Visual Environments and Games*, pages 135–140, Qinhuangdao, Hebei, China, October 2011.

[25] Vicki Bruce, Mark A. Georgeson, and Patrick R. Green. *Visual Perception: Physiology, Psychology and Ecology*. Psychology Press, 2003.

[26] K.I. Calvert, M.B. Doar, and E.W. Zegura. Modeling Internet topology. *Communications Magazine, IEEE*, 35(6):160–163, June 1997.

[27] Renato Capocelli, Luisa Gargano, and Ugo Vaccaro. Time bounds for broadcasting in bounded degree graphs. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 19–33. Springer Berlin / Heidelberg, 1990. 10.1007/3-540-52292-1_2.

[28] M.G. Ceruti, T.L. Wright, B.J. Powers, and S.C. McGirr. Data pedigree and strategies for dynamic level-one sensor data fusion. In *Proceedings of the 9th International Conference on Information Fusion*, July 2006.

[29] K.C. Chang, Chee-Yee Chong, and S. Mori. On scalable distributed sensor fusion. In *Proceedings of the 11th International Conference on Information Fusion*, July 2008.

[30] Kou-Chu Chang. Autonomous and scalable data fusion and dissemination in netcentric world. In *Proceedings of the 11th International Conference on Information Fusion*, July 2008.

[31] Rahul Chaudhari, Clemens Schuwerk, Verena Nitsch, Eckehard Steinbach, and Berthold Faerber. Opening the haptic loop: Network degradation limits for haptic task performance. In *HAVE '11: Proceedings of the IEEE International Symposium on Haptic Audio-Visual Environments and Games*, pages 56–61, Qinhuangdao, Hebei, China, October 2011.

[32] C.Y. Chong, S. Mori, and K.C. Chang. Distributed multitarget multisensor tracking. *Multitarget-multisensor tracking: Advanced applications*, 1:247–295, 1990.

[33] Po-Jen Chuang, Bo-Yi Li, and Tun-Hao Chao. Hypercube-based data gathering in wireless sensor networks. *J. Inf. Sci. Eng.*, 23(4):1155–1170, 2007.

[34] D. Clark, S. Julier, R. Mahler, and B. Ristic. Robust multi-object sensor fusion with unknown correlations. In *Sensor Signal Processing for Defence*, 2010.

[35] William S. Cleveland. LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression. *The American Statistician*, 35(1), 1981.

[36] Darrell Conklin. Structured concept discovery: Theory and methods. Technical Report 94-366, Queen's University, Kingston, Ontario, Canada K7L 3N6, June 1994.

[37] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[38] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *SIGCOMM*, pages 15–26. ACM, 2004.

[39] G.B. Dantzig. *Maximization of a Linear Function of Variables Subject to Linear Inequalities*, chapter XXI, pages 339–347. Number 13. John Wiley and Sons, Inc., 1951.

[40] George B. Dantzig and Mukund N. Thapa. *Linear programming 1: Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[41] Ton de Jong and Wouter R. van Joolingen. Scientific discovery learning with computer simulations of conceptual domains. *Review of educational research*, 68:179–202, 1998.

[42] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification, December 1998.

[43] Michael J. Dinneen, Geoffrey Pritchard, and Mark C. Wilson. Degree- and time-constrained broadcast networks. *Networks*, 39(3):121–129, 2002.

[44] Marcel Dischinger, Andreas Haeberlen, Ivan Beschastnikh, Krishna P. Gummadi, and Stefan Saroiu. Satellitelab: Adding heterogeneity to planetary-scale network testbeds. *SIGCOMM Comput. Commun. Rev.*, 38(4):315–326, 2008.

[45] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 43–56, New York, NY, USA, 2007. ACM.

[46] J. Eberspächer, H.J. Vögel, C. Bettstetter, and C. Hartmann. *GSM - Architecture, Protocols and Services.* John Wiley & Sons, 3rd edition, 2009.

[47] Ulrich W. Eisenecker, Frank Blinn, and Krzysztof Czarnecki. A solution to the constructor-problem of mixin-based programming in C++. In *Proceedings of the GCSE '00 Workshop on C++ Template Programming at the Second International Symposium on Generative and Component-Based Software Engineering*, Erfurt, Germany, October 2000.

[48] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, March 1975.

[49] O. Erdinc, P. Willett, and Y. Bar-Shalom. Probability hypothesis density filter for multitarget multisensor tracking. In *Proceedings of the 8th International Conference on Information Fusion*, volume 1, pages 1–8, July 2005.

[50] S. Even and B. Monien. On the number of rounds necessary to disseminate information. In *Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '89, pages 318–327, New York, NY, USA, 1989. ACM.

[51] Vincenzo De Florio and Chris Blondia. Robust and tuneable family of gossiping algorithms. In *PDP '12: Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 154–161, Garching, Germany, February 2012. IEEE Comp. Soc.

[52] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *Proceedings of the USENIX '05 Annual Technical Conference*, ATEC '05, pages 13–13, Berkeley, CA, USA, 2005. USENIX Association.

[53] Geoffrey C. Fox, Roy D. Williams, and Paul C. Messina. *Parallel Computing Works!* Parallel processing scientific computing. Morgan Kaufmann, 1994.

[54] John F. Gantz, Christopher Chute, Alex Manfrediz, Stephen Minton, David Reinsel, and Wolfgang Schlichting Anna Toncheva. The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011. White paper, International Data Corporation, 5 Speen Street, Framingham, MA 01701, 2008.

[55] A.P. Garcia, J. Oliver, and D. Gosch. An intelligent agent-based distributed architecture for smart-grid integrated network management. In *LCN '10: Proceedings of the 35th IEEE Conference on Local Computer Networks*, pages 1013–1018, October 2010.

[56] J. Gray, A. Bosworth, A. Lyaman, and H. Pirahesh. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 152–159. IEEE Computer Society, March 1996.

[57] S. Grime and H.F. Durrant-Whyte. Data fusion in decentralized sensor networks. *Control engineering practice*, 2(5):849–863, 1994.

[58] Xiaohui Gu, Zhen Wen, Philip S. Yu, and Zon-Yin Shae. Supporting multi-party voice-over-IP services with peer-to-peer stream processing. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, pages 303–306, New York, NY, USA, 2005. ACM Press.

[59] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proceedings of the SIGCOMM Internet Measurement Workshop 2002*, 2002.

[60] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2012.

[61] John P. Hayes, Trevor N. Mudge, Quentin F. Stout, Stephen Colley, and John Palmer. Architecture of a hypercube supercomputer. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 653–660, University Park, PA, USA, August 1986.

[62] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast fourier transform. *ASSP Magazine, IEEE*, 1(4):14–21, October 1984.

[63] José Hernández, Iain W. Phillips, and Javier M. Moguerza. An SS-SVM approach to generate synthetic network delays, 2005.

[64] G. Hooghiemstra and P. Van Mieghem. Delay distributions on fixed internet paths. *Delft University of Technology, report20011031*, 2001.

[65] Juraj Hromkovic. *Dissemination of information in communication networks: broadcasting, gossiping, leader election, and fault-tolerance*. Texts in theoretical computer science. Springer, 2005.

[66] Juraj Hromkovic, Ralf Klasing, Burkhard Monien, and Regine Peine. Dissemination of information in interconnection networks (broadcasting & gossiping), 1996.

[67] M.B. Hurley. An information theoretic justification for covariance intersection and its generalization. In *Proceedings of the 5th International Conference on Information Fusion*, volume 1, pages 505–511. IEEE, 2002.

[68] Information Sciences Institute. Internet protocol - darpa internet program protocol specification. Technical Report rfc791, Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, September 1981.

[69] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile computing and Networking*, pages 56–67, New York, NY, USA, 2000. ACM.

[70] ITU. *Integrated services digital network (ITU-T Recommendation G.705)*. International Telecommunications Union, November 1980.

[71] ITU. *Integrated services digital network (ITU-T Recommendation I Series)*. International Telecommunications Union, November 1984.

[72] W. Jiang. Detecting and measuring asymmetric links in an IP network, 1999.

[73] Hai Jin, Fei Luo, Qin Zhang, Xiaofei Liao, and Hao Zhang. GTapestry: A locality-aware overlay network for high performance computing. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 76–81, June 2006.

[74] S.J. Julier and J.K. Uhlmann. A non-divergent estimation algorithm in the presence of unknown correlations. In *Proceedings of the 1997 American Control Conference*, volume 4, pages 2369–2373, June 1997.

[75] M. Kallstrom and S.S. Thakkar. Programming three parallel computers. *Software, IEEE*, 5(1):11–22, January 1988.

[76] Yasusi Kanada. Multi-context voice communication controlled by using an auditory virtual space. In *CCN '04: Proceedings of the 2nd International Conference on Communication and Computer Networks*, November 2004.

[77] H.P. Katseff. Incomplete hypercubes. *IEEE Transactions on Computers*, 37(5):604–608, May 1988.

[78] Sebastian Kaune, Konstantin Pussep, Christof Leng, Aleksandra Kovacevic, Gareth Tyson, and Ralf Steinmetz. Modelling the Internet delay space based on geographical locations. In *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 301–310, Washington, DC, USA, 2009. IEEE Computer Society.

[79] Thomas Keating, Michael Barnett, Sasha A. Barab, and Kenneth E. Hay. The virtual solar system project: Developing conceptual understanding of astronomical concepts through building three-dimensional computational models. *Journal of Science Education and Technology*, 11(3):261–275, 2002.

[80] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[81] Kitz - ADSL Broadband Information Site. Interleaving explained. `http://www.kitz.co.uk/adsl/interleaving.htm`, June 2012. Retrieved 2012-07-11.

[82] Ralf Klasing, Burkhard Monien, Regine Peine, and Elena A. Stöhr. Broadcasting in butterfly and DeBruijn networks, 1992.

[83] D.E. Knuth. *Selected papers on computer science*, chapter 11, pages 194–200. CSLI lecture notes. CSLI Publications, 1996.

[84] Robert Kozma. *Innovations in Science and Mathematics Education: Advanced Designs for Technologies of Learning*, chapter The Use of Multiple Representations and the Social Construction of Understanding in Chemistry, pages 11–46. Erlbaum Associates, Mahwah, NJ, 2000.

[85] David Krumme. Fast gossiping for the hypercube. *SIAM Journal on Computing*, 21:365–380, 1992.

[86] David W. Krumme, George Cybenko, and K. N. Venkataraman. Gossiping in minimal time. *SIAM Journal on Computing*, 21:111–139, 1992.

[87] H.T. Kung and Carnegie-Mellon University. Computer Science Dept. *Let's design algorithms for VLSI systems.* Computer Science Dept., Carnegie-Mellon University, 1979.

[88] Gerald Kunzmann. Recursive or iterative routing? Hybrid! In Paul Müller, Reinhard Gotzhein, and Jens B. Schmitt, editors, *GI '05: Kommunikation in Verteilten Systemen (KiVS), Kurzbeiträge und Workshop der 14. GI/ITG-Fachtagung*, volume P-61 (2005) of *Lecture Notes in Informatics*, pages 189–192. GI, Gesellschaft für Informatik, Bonn, 2005.

[89] David Kushner. Engineering everquest: online gaming demands heavyweight data centers. *Spectrum, IEEE*, 42(7):34–39, July 2005.

[90] Alexey Kuznetsov and YOSHIFUJI Hideaki. iputils: Small useful utilities for Linux networking. http://www.skbuff.net/iputils/.

[91] Jaroslav Kysela, Abramo Bagnara, Takashi Iwai, and Frank van de Pol. The advanced Linux sound architecture (ALSA) C library reference. `http://www.alsa-project.org/alsa-doc/alsa-lib/index.html`, January 2012.

[92] F.T. Leighton and B. Monien. *Einführung in parallele Algorithmen und Architekturen: Gitter, Bäume und Hypercubes.* Internat. Thomson Publ., 1st edition, 1997.

[93] Jonathan Lennox and Henning Schulzrinne. A protocol for reliable decentralized conferencing. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 72–81, New York, NY, USA, 2003. ACM Press.

[94] Jin Li. Mutualcast: A serverless peer-to-peer multiparty real-time audio conferencing system. In *ICME '05: IEEE International Conference on Multimedia and Expo*, pages 602–605, July 2005.

[95] Xiaozhou Li and C. Greg Plaxton. On name resolution in peer-to-peer networks. In *POMC '02: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*, pages 82–89, New York, NY, USA, 2002. ACM.

[96] Arthur Liestman and Dana Richards. An introduction to perpetual gossiping. In K. Ng, P. Raghavan, N. Balasubramanian, and F. Chin, editors, *Algorithms and Computation*, volume 762 of *Lecture Notes in Computer Science*, pages 259–266. Springer Berlin / Heidelberg, 1993. 10.1007/3-540-57568-5_256.

[97] Huaiyu Liu and Simon S. Lam. Neighbor table construction and update for resilient hypercube routing in P2P networks. Technical Report TR-07-31, July 2007.

[98] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36:131–146, December 2002.

[99] Damien Magoni and Jean-Jacques Pansiot. Analysis and comparison of Internet topology generators. In *Proceedings of the Second International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications (NETWORKING)*, pages 364–375, London, UK, 2002. Springer-Verlag.

[100] Damien Magoni and Jean-Jacques Pansiot. Internet topology modeler based on map sampling. In *ISCC '02: Proceedings of the 7th International Symposium on Computers and Communications*, page 1021, Washington, DC, USA, 2002. IEEE Computer Society.

[101] R.P.S. Mahler. Optimal/robust distributed data fusion: A unified approach. In *Proceedings of SPIE*, volume 4052, page 128, 2000.

[102] R.P.S. Mahler. Multitarget Bayes filtering via first-order multitarget moments. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1152–1178, October 2003.

[103] R.P.S. Mahler. "Statistics 101" for multisensor, multitarget data fusion. *Aerospace and Electronic Systems Magazine, IEEE*, 19(1):53–64, January 2004.

[104] R.P.S Mahler. *Statistical multisource-multitarget information fusion*. Artech House, 2007.

[105] Gregor Maier. *Residential Broadband Internet Traffic: Characterization and Security Analysis*. PhD thesis, Technische Universität Berlin, Berlin, September 2010.

[106] A. Makarenko, A. Brooks, T. Kaupp, H. Durrant-Whyte, and F. Dellaert. Decentralised data fusion: A graphical model approach. In *Proceedings of the 12th International Conference on Information Fusion*, pages 545–554, July 2009.

[107] Q.M. Malluhi and M.A. Bayoumi. The hierarchical hypercube: A new interconnection topology for massively parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):17–30, January 1994.

[108] Ramón A. Mata-Toledo and Matthew A. Willis. Visualization of random sequences using the chaos game algorithm. *J. Syst. Softw.*, 39:3–6, October 1997.

[109] Ernst Mayr and Ralph Werchner. Divide-and-conquer algorithms on the hypercube. In P. Enjalbert, A. Finkel, and K. Wagner, editors, *STACS 93*, volume 665 of *Lecture Notes in Computer Science*, pages 153–162. Springer Berlin / Heidelberg, 1993. 10.1007/3-540-56503-5_18.

[110] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *SIGCOMM Comput. Commun. Rev.*, 30(2):18–28, 2000.

[111] Michel Mollard. Two characterizations of generalized hypercube. *Discrete Mathematics*, 93(1):63–74, 1991.

[112] Jerry Morrison. "EA IFF 85" standard for interchange format files. `http://www.martinreddy.net/gfx/2d/IFF.txt`, January 1985. (Proposed Standard).

[113] Amarnath Mukherjee. On the dynamics and significance of low frequency components of Internet load. *Internetworking: Research and Experience*, 5:163–205, 1992.

[114] T. S. Eugene Ng and Hui Zhang. Towards global network positioning. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 25–29, 2001.

[115] T. S. Eugene Ng and Hui Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM '02: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179, 2002.

[116] Pedro Novo. *Communication Acoustics*, chapter Auditory Virtual Environments, pages 277–292. Springer, 1st edition, July 2005.

[117] L.-L. Ong, T. Bailey, H. Durrant-Whyte, and B. Upcroft. Decentralised particle filtering for multiple target tracking in wireless sensor networks. In *Proceedings of the 11th International Conference on Information Fusion*, pages 1–8, July 2008.

[118] Jianli Pan, R. Jain, S. Paul, M. Bowman, Xiaohu Xu, and Shanzhi Chen. Enhanced MILSA architecture for naming, addressing, routing and security issues in the next generation internet. In *ICC '09: Proceedings of the IEEE International Conference on Communications*, pages 1–6, June 2009.

[119] Peerfactsim.kom: A large scale simulation framework for peer-to-peer systems. http://peerfact.kom.e-technik.tu-darmstadt.de/.

[120] John Durham Peters. *Communication as ...: Perspectives on Theory*, chapter Communication as Dissemination, pages 211–222. Thousand Oaks: Sage Publications, 2005.

[121] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, New York, 1997. ACM.

[122] Ithiel de Sola Pool. *Forecasting the Telephone: A Retrospective Technology Assessment of the Telephone*. Ablex Publishing, January 1982.

[123] Franco P. Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, 24(5):300–309, May 1981.

[124] Bruno Quoitin and Steve Uhlig. Modeling the routing of an autonomous system with CBGP. *IEEE Network Magazine*, 19, 2005.

[125] Martin Raack, Dominic Battré, André Höing, and Odej Kao. Papnet: A proximity-aware alphanumeric overlay supporting ganesan on-line load balancing. In *ICPADS '09: Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems*, pages 440–447, Washington, DC, USA, 2009. IEEE Computer Society.

[126] Milena Radenkovic and Chris Greenhalgh. Multi-party distributed audio service with TCP fairness. In *Proceedings of the 10th ACM International Conference on Multimedia*, pages 11–20, New York, NY, USA, 2002. ACM.

[127] Milena Radenkovic, Chris Greenhalgh, and Steve Benford. Deployment issues for multi-user audio support in cves. In *VRST '02: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 179–185, New York, NY, USA, 2002. ACM.

[128] Statistical analysis of random numbers. http://www.random.org/analysis/.

[129] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[130] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, pages 99–100, August 2001.

[131] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-45518-3_18.

[132] G. Schiele, R. Süselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis. Requirements of peer-to-peer-based massively multiplayer online gaming. In *CCGRID '07: Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid*, pages 773–782. IEEE, 2007.

[133] Gregor Schiele, Shun-Yun Hu, Daniel Weiskopf, and Ben Leong. Challenges in designing massively multiuser virtual environments: Experiences from MMVE2008. *Intl. Journal on Advanced Media and Communication*, 2(4):325–330, 2008.

[134] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup – shaping up peer-to-peer networks, 2002.

[135] Andre Schuchardt. *Eine kurze Geschichte der Sumerer (und Akkader)*. GRIN Verlag GmbH, 2010.

[136] Charles L. Seitz. The cosmic cube. *Communincations of the ACM*, 28(1):22–33, January 1985.

[137] Xiaohong Sheng, Yu-Hen Hu, and Parameswaran Ramanathan. Distributed particle filter with GMM approximation for multiple targets localization and tracking in wireless sensor network. In *IPSN '05: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, Piscataway, NJ, USA, 2005. IEEE Press.

[138] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 296–305, New York, NY, USA, 2001. ACM.

[139] Kurt D. Squire. *Replaying History: Learning World History through Playing Civilization III*. PhD thesis, Indiana University, January 2004.

[140] Swing (java foundation classes). `http://download.oracle.com/javase/6/docs/technotes/guides/swing/`.

[141] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, 4th edition, 2003.

[142] Daniel Terdiman. 'Second Life': Don't worry, we can scale. `http://news.cnet.com/2100-1043_3-6080186.html`, June 2006.

[143] P.R. Thompson and H. Durrant-Whyte. Decentralised data fusion in 2-tree sensor networks. In *Proceedings of the 13th Conference on Information Fusion*, pages 1–8, July 2010.

[144] Murat Uney, Simon Julier, Daniel Clark, and Branko Ristic. Monte Carlo realisation of a distributed multi-object fusion algorithm. In *SSPD '10: Sensor Signal Processing for Defence 2010*, 2010.

[145] András Varga. OMNeT++ — Portable simulation environment in C++. In *Proceedings of the Annual Students' Scientific Conference (TDK)*. Technical University of Budapest, 1992.

[146] B.-N. Vo, S. Singh, and A. Doucet. Sequential Monte Carlo methods for multitarget filtering with random finite sets. *IEEE Transactions on Aerospace and Electronic Systems*, 41(4):1224–1245, October 2005.

[147] B.-T. Vo, B.-N. Vo, and A. Cantoni. The cardinalized probability hypothesis density filter for linear Gaussian multi-target models. In *40th Annual Conference on Information Sciences and Systems*, pages 681–686, March 2006.

[148] Ba-Ngu Vo and Wing-Kin Ma. The Gaussian mixture probability hypothesis density filter. *Signal Processing, IEEE Transactions on*, 54(11):4091–4104, November 2006.

[149] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.

[150] Hsin-Kai Wu, Joseph S. Krajcik, and Elliot Soloway. Promoting conceptual understanding of chemical representations: Students use of a visualization tool in the classroom, 2001.

[151] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 5–14, 2002.

[152] Keiichi Yasumoto and Klara Nahrstedt. Ravitas: Realistic voice chat framework for cooperative virtual spaces, 2005.

[153] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.