Application-level Simulation for Network Security

Stephan Schmidt
Rainer Bye
Joël Chinnow
Karsten Bsufka
Ahmet Camtepe
Sahin Albayrak
DAI-Labor, Berlin Institute of Technology,
Ernst-Reuter-Platz 7, 10587 Berlin,
Germany
nessi@dai-labor.de

NeSSi (network security simulator) is a novel network simulation tool which incorporates a variety of features relevant to network security distinguishing it from general-purpose network simulators. Its capabilities such as profile-based automated attack generation, traffic analysis and support for detection algorithm plug-ins allow it to be used for security research and evaluation purposes. *NeSSi* has been successfully used for testing intrusion detection algorithms, conducting network security analysis and developing overlay security frameworks. *NeSSi* is built upon the agent framework *JIAC*, resulting in a distributed and extensible architecture. In this paper, we provide an overview of the *NeSSi* architecture as well as its distinguishing features and briefly demonstrate its application to current security research projects.

Keywords: network simulation, network security, discrete-event simulation, packet-level simulation, application-level simulation

1. Introduction

In contemporary communication infrastructures, IP-based computer networks play a prominent role. The deployment of these networks is progressing at an exponential rate as different kinds of participants such as corporations, public authorities and individuals rely on sophisticated and complex services and communication systems. With regard to information security, this leads to new challenges as large amounts of data, which may hold malicious content such as worms, viruses or Trojans, are transferred over open networks. Network security measures dealing with these threats can be implemented in the network itself as well as at hosts connected to access routers of the network. The host-based approach has its merits, especially with respect to the scalability of a resulting security framework; for example, placing security capabilities such as firewalls or virus scanners on individual hosts does not inhibit the traffic traveling through the network. However, as the hosts are generally not under the control of network operators, there is no way of ensuring a certain network-wide security policy.

A consequence for network service providers (NSPs) striving to offer improved security features to their customers as a value-adding feature is to devise a security framework in which detection devices are placed within the network. Before doing so, the NSP must take into account that it is not desirable to make frequent changes or experiment with various security feature deployments in the network infrastructure of a production system. For this reason, network operators can greatly profit from a network simulation tool in which various features of the security architectures can be tested in order to ensure maximum attack detection efficiency before the actual physical deployment. The advantage over conventional testbeds is

SIMULATION, Vol. 86, Issue 5-6, May-June 2010 311–330

DOI: 10.1177/0037549709340730
Figures 1–12 appear in color online: http://sim.sagepub.com

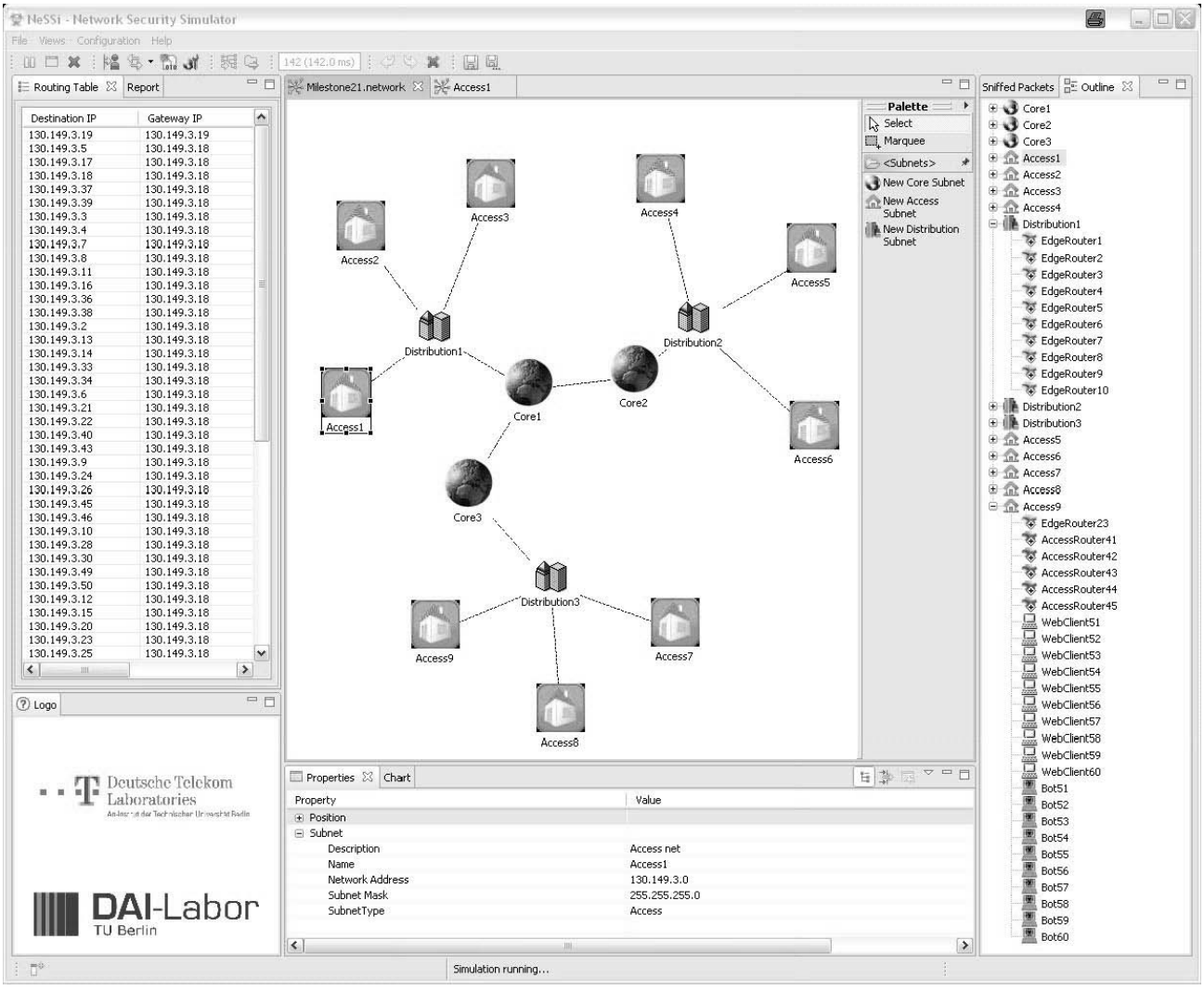

Figure 1. GUI of *NeSSi*: main *Network Editor* window

the low cost and ease at which tests can be carried out. The presented network security simulator *NeSSi* (see Figure 1) allows NSPs to experiment with different network-centric security framework setups and algorithms in order to evaluate and compare intrusion detection efficiency and operational costs. In this fashion, a devised security framework can be tested in a simulation environment before the actual detection units are physically deployed in the network.

In the following section, we provide an overview of existing network simulation tools, focusing on security evaluation capabilities. We then describe the general software architecture of *NeSSi* in Section 3. Subsequently, we focus on the traffic and protocol support in Section 4 and security features in Section 5 and finally demonstrate how it can be used for the setup and execution of realistic attack scenarios. Our conclusions and outlook on future extensions of *NeSSi* are given in Section 7.

2. Related Work

In recent years, the research community has used various network simulation tools for the verification of new algorithms, the investigation of design and interaction behavior of newly developed protocols as well as the examination of performance issues encountered in large-scale network architectures. The most popular general-purpose software tool in the research community is the open-source network simulator *ns2* [1]. *ns2* performs network simulation using a discrete-event model. This approach has several advantages regarding application performance and scalability. These important issues are discussed, for example, in [2] and [3]. Discrete simulation allows very cost-efficient exploration and experimentation with real-life network topologies and architectures. In many areas involving network analysis, *ns2* is a powerful tool; how-

ever, it also poses certain limitations to the user. Concerning the aspect of simulation efficiency for large networks, *ns2* does not support out-of-the-box a parallel execution model. It does not support generic sockets, but protocols have to be rewritten for use in *ns2*. Moreover, the script language interface is not intuitive for use by novice users. Alternatives to *ns2* are, for example, the *QualNET* simulator [4] or the Georgia Tech Network Simulator *GTNetS* [5], which, in contrast to *ns2*, also offers parallel execution support for large-scale simulation.

Most important of all, these simulators have limitations in standard support of real-world network security evaluation. A lot of work has been performed in the limited area of worm simulation, as described extensively by Wei et al. [6]. In this work, the authors noted that existing simulators are mostly single-machine tools and hence do not scale to model realistic attack mechanisms in large-scale real-world networks. They proposed a distributed approach termed *PAWS*, which is nevertheless not a comprehensive tool but limited to worm simulation only. Another security evaluation tool is *RINSE*, which is described by Liljenstam et al. in [7]. It focuses on supporting real-time large-scale simulations and allows for realistic emulation of CPU and memory effects within the simulation. However, there is no mention of application-level simulation capabilities in *RINSE*. The same drawback exists in the solution presented in [8], although it allows model-driven attack tree-based simulation in a reusable object-oriented software architecture.

2.1 Our Contribution

NeSSi aims to fill this gap by offering simulation and evaluation features specifically tailored to meet the needs of security experts and network administrators. This target group will be able to test and evaluate the performance of commonly available security solutions as well as new research approaches. As a distinguishing feature to the previous tools described above, *NeSSi* provides extensive support of complex application-level scenarios on top of a faithful simulation of the TCP/IP protocol stack. Simulated networks are modeled to reflect real-world network topologies by supporting subnet-layer modeling and different node types with varying capabilities (core and access subnets, wireless networks etc.). In particular, *NeSSi* adheres to a modular design pattern and fosters the integration of third-party applications via a standardized plug-in interface. Furthermore, it provides a comprehensive detection API for the integration and evaluation of simulated as well as external detection units. In particular, special common attack scenarios are supported and can be simulated, worm-spread scenarios and botnet-based distributed denial-of-service (DdoS) attacks are only two of the example scenarios supported by *NeSSi*. In addition, customized profiles expressing the node behavior can be applied within the simulation.

The application layer simulation capabilities in *NeSSi* are provided by distributed software agents, introducing another layer of abstraction. In order to maintain scalability, a parallel execution model is used in conjunction with a discrete-event model. In this context, the agent platforms are running on multiple parallel machines and connect independently to a database server from which simulation results can be retrieved in an asynchronous and concurrent fashion. The graphical user interface (GUI) allows for real-time inspection and configuration of scenarios.

3. NeSSi Architecture

The design of a packet-level discrete-event simulator is a challenging task. In order to handle the inherent complexity, *NeSSi* has been structured into three distinct components, the GUI, the simulation backend and the result database. Each of these modules may be run on separate machines depending on the computational requirements; furthermore, this modular design facilitates network security researchers using *NeSSi* to easily exchange network topologies, scenario definitions and simulation results via remote repositories. In the following sections, we describe each of these modules in turn and delineate the workflow in *NeSSi*, beginning with the creation of a network from scratch up to the analysis of the simulation results.

3.1 GUI

The GUI is a rich client platform (RCP) application based on the *Eclipse* framework [9]. It uses the *Standard Widget Toolkit* (SWT) [10], a cross-platform open-source widget toolkit which allows *NeSSi* to be run on almost all operating systems. As a RCP application, *NeSSi* is structured into *views*, *editors* and *perspectives*.

A view is a composite widget for displaying data of a certain type to the user; the different views implemented in *NeSSi* will be described later in this section. Editor windows contain the main data of interest (in the case of *NeSSi* graphical representations of networks respectively subnets), while the surrounding views display context-related information based on the selection in the editor window. Perspectives on the other hand are used to bundle thematically related views and editors to present the user with an interface to execute the desired task while hiding functionality that is not essential to the current task.

The graphical frontend of *NeSSi* allows the user to create and edit network topologies, attach runtime information, and schedule them for execution at the simulation backend. On the other hand, finished (or even currently executing, long-running) simulations can be retrieved from the database server and the corresponding simulation results are visualized in the GUI. This constitutes two distinct use cases: pre- and post-simulation visualization. For this reason, the GUI has been divided in two perspectives, a *Network Editor* perspective and a *Network Simulation* perspective.

3.1.1 Network Editor Perspective

The *Network Editor* perspective comprises a main *Network Editor* window grouped with a variety of different views which provide additional information pertaining to the element currently selected in the *Network Editor*. Some of these views appear in both perspectives since they are valuable in the editing as well as the evaluation phase. These are as follows.

Console. Network status information and logged events are displayed here. The desired level of verbosity can be configured.

Routing table. When a network node, i.e. a client, server or router, is selected, the respective routing table is displayed. For each reachable target machine, it contains the IP addresses of the gateway, the subnet mask and the hop count.

Properties. Displays and allows a number of properties to be edited for the selected element. Among other information, device name and device type are displayed for nodes; for links, bandwidth, latency and maximum transfer unit (MTU) is available.

Statistics. Graphical representation of simulation results. The content of this view adapts to the element selected in the editor or the outline.

Apart from these, by reusing the existing plug-ins available for Eclipse, we offer valuable standard functionality in *NeSSi* such as the following.

Team support. Users of *NeSSi* can share networks, scenarios and sessions as well as database resources via remote repositories based on CVS or SVN.

Update functionality. New versions can be downloaded via an update site. The updates are deployed as separate features, allowing the user to decide which components of their *NeSSi* distribution they want to update. *NeSSi* is open source since late 2008; the update site location is published at http://nessi2.de.

Integrated help. *NeSSi* offers an online help system. This comprises different types of manuals (user, developer, etc.), Javadoc documentation and context-based help for the current UI selection.

Other views such as *Bookmarks*, *Tasks* and *Progress* are also supported and used by *NeSSi*. They are not described here for the sake of brevity.

Before starting to create their first simulation, the user needs to become familiarized with the network/scenario/session concept which *NeSSi* is built upon. This paradigm serves to distinguish static topology information from dynamic behavioral information with the goal of enhancing the reusability of previously created elements. Simulations often need to be carried out for large networks which differ only in a few parameters while most of the information stays the same; sometimes the user wants to apply different traffic-generation functionality in the same network without having to create and subsequently manage unnecessary copies of the same static topology information.

Network topology. The network topology describes the static information contained in the network, i.e. the nodes of the network, their (initial) properties and how they are interconnected. Thus, a topology can be reused in combination with several different scenarios.

Scenario. A scenario contains runtime information attached to a particular network topology. For instance, a scenario may contain the traffic-generation properties for different nodes in the network. A scenario can be reused in combination with different sessions. For detailed information on the type of scenarios that can be created with the standard *NeSSi* version, refer to Section 4.

Session. Finally, the session information contains additional information specific for the selected scenario. For example, the user may be interested only in a certain type of traffic, or the scenario should be carried out repeatedly, but with different tick runtimes (*tick* is the term for the atomic discrete time unit). This kind of information is encapsulated in the Session object for a particular scenario.

Network topologies can be created by choosing network elements such as routers and different kinds of end devices such as web clients, mail servers, etc., and adding them from a palette to the network editor main window via drag-and-drop (cf. Figure 1). In the simulation, agents realize the behavior of the nodes (cf. Section 3.2.2). The node modeling occurs in two layers; the first layer reflects their role in the network (client, server, switch or router). This is further refined according to their application-level role (such as web clients, mail clients, IRC server, etc.) and inherent functionality.

The network is hierarchically structured by dividing it into subnets. For example, a large-scale network usually consists of a core area, a number of distribution networks (for example, university or metropolitan area networks) and access networks, i.e. company networks. Such a network with a hierarchical structure is displayed in the central network editor window of Figure 1. It is also possible to automatically generate networks by specifying various parameters such as the number of individual subnets and their types, node degree, topology (star, ring, etc.), average link bandwidth in the core, distribution and access subnets and many more (see Figure 2). This is especially useful for generating large-scale topologies, where the network generation is a two-step process: the user generates a topology automatically and adjusts the resulting network by

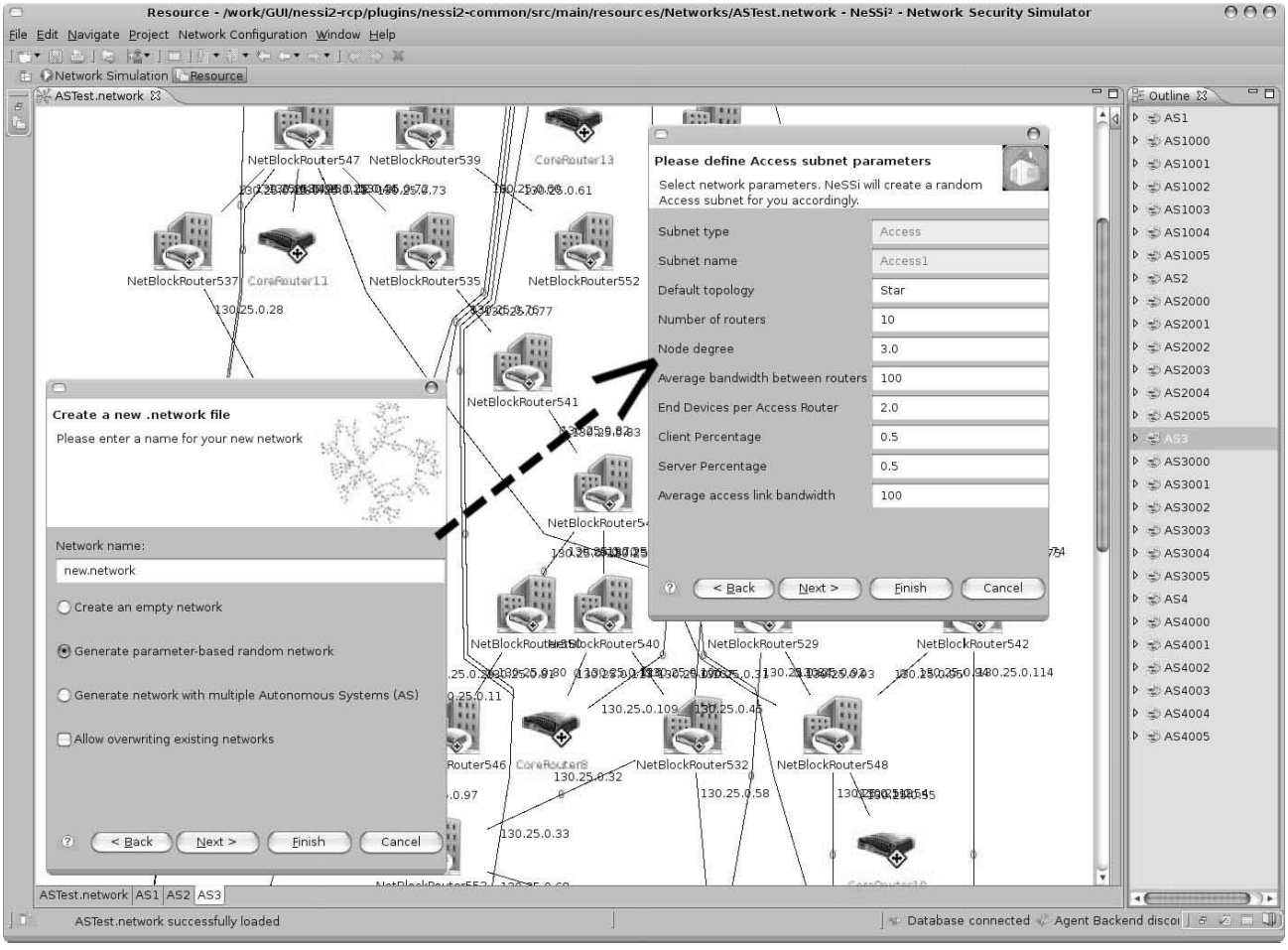

Figure 2. Wizard-assisted network creation

fine-tuning individual devices or device properties. These subnets and their individual network elements such as routers and links and their properties, i.e. routing tables, network interfaces, etc., are modeled using the Eclipse Modeling Framework (EMF) [11] which also allows automated source-code generation. Consequently, the model can easily be extended to include new features or adapted to match new requirements. Supported standard device properties are processing speed, packet queuing mechanisms and supported routing protocols; link properties include delay and bandwidth.

Individual subnets are connected at edge routers, which are logically assigned to both subnets. The user can jump to the corresponding other subnet by double-clicking on an edge router, which allows for easy navigation through the entire network, although individual subnets are displayed in separate tabs.

The user interacts with the application via context-sensitive actions. For more complex operations, the user is supported by wizards. For example, when an IRC server is drag-and-dropped on the network editor, a context-menu action is dynamically created for this node which allows the execution of a pre-defined IRC exploit scenario. When the user selects it, a wizard appears, describing the scenario and guiding the user through the process.

After the static network topology configuration is complete, the dynamic scenario components need to be created. A scenario comprises elementary building blocks for each device in the network, the concept of node *profiles*. Node profiles allow the customization of network node behavior in order to automatically generate traffic adhering to well-defined characteristics, simulate network outages by means of router or link failures as well as evaluating network-based defense measures. The basic elements used to describe these *profiles* are *actions* and *plans*.

Actions. Actions are clustered into three different types: traffic, failure and detection units. Traffic actions describe the behavior of a single service (HTTP, SMTP, etc.) with various parameters depending on

their type. Failure actions are used to simulate network failures such as link outage or router malfunction. Detection unit actions configure security-related applications which utilize the *Detection Unit API* (see Section 5.2).

Plans. The time characteristics are described with plans for the corresponding action. They contain at least one time span in which the action is executed according to various distribution functions (standard, binomial, etc.).

Profiles. Profiles, the top-level elements, describe the actual node behavior by utilizing different plans. An execution plan can be included in a profile multiple times for repeated execution. After defining a profile, it can easily be attached to several clients by enabling it in the respective clients' context menu. In addition, the profiles can be distributed in a percentage-based fashion on end devices, routers or (in the case of failures) links. Security-related applications such as network intrusion detection system (NIDS) sensor components can also be deployed on a selection of machines based on a user-defined measure of optimality in order to maximize the efficiency of the NIDS. To this end, the set of machines selected for running the profile may, for example, be obtained by a betweenness centrality or game-theory approach (cf. Section 6.2).

The sum of all profiles for a given network topology constitutes the *scenario*. In order to execute a scenario, the last set of parameters to specify are the *session* parameters. These include, for example, the number of times a given scenario is to be executed and the types of events to log in the database. The security expert may wish to run a single scenario multiple times if the employed node profiles are not statically configured but make use of distribution functions for generating network traffic. In this case, multiple runs allow the generation of statistically significant mean values and the comparison of individual runs to examine standard deviations.

The ability to specify the types of events to log allows for a higher degree of scalability for large-scale simulations, where the user is only interested in a particular subset of the simulation results. For example, it can be configured that only application-level traffic for a certain protocol type is to be logged, which may greatly increase the simulation speed when a lot of other background traffic is generated which does not need to be logged.

Supplementary *NeSSi* includes automated attack generation for the purpose of examining security-related network features. Several adversary models are supported, among others worm spread and DDoS attack models. This feature has proven highly useful in the research of the cooperative detection approach for generating simulation data (cf. Section 5). In the resulting paper [12], traffic statistics of captured real traffic data were mapped to node profiles.

Once a session has been created for a particular scenario and topology, it can be transmitted to the simulation backend for execution. For details on how the simulation is carried out refer to Section 3.2; in the following section we first describe how the results are presented to the user of NeSSi.

3.1.2 Network Simulation Perspective

The *Network Simulation* perspective contains a set of related views which allow the visualization of simulation results. Upon connecting to the database server, the available simulation results are depicted in a tree structure annotated with the date of the execution and the user who has submitted them. The desired session can be selected, and the corresponding simulation results are downloaded for display.

The user has several options for displaying the simulation results. The main editor window contains the network topology associated with the selected session; naturally however, no network elements may be edited or deleted (in contrast to the *Network Editor* perspective). In the menu bar, the user can step forward and backward through the simulation to inspect the state of the network at the desired tick. For demonstration purposes, a *playback* functionality is also provided where the simulation is visualized at the desired speed.

In the main network window, the status of the network elements is visualized by different means. For example, link widths and colors change depending on the type and amount of traffic on the link, while devices which have failed (for example, due to a successful attack) appear grayed out, or devices which are infected but operational are flagged with a corresponding symbol. Since not all relevant information can be displayed in the editor window, however, two additional views contain context-based information relevant to the network element selected in the editor. The *Properties* view displays detail information for the current tick (for example, routing tables may change over time), while the *Statistics* view contains graphical information of logged events for the selected network element. For example, this may be a graphical representation of the packets on a given link, sorted by protocol type. As an added feature, this view is highly configurable and may be adapted to show cumulative values, restrict the results to certain subsets of interesting data sources and so on.

3.2 Simulation Backend

The actual simulation is performed on a machine with hardware dedicated solely to this purpose, the *simulation backend*. At the Berlin Institute of Technology, for example, the *NeSSi* simulation backend runs on a Sun XFire

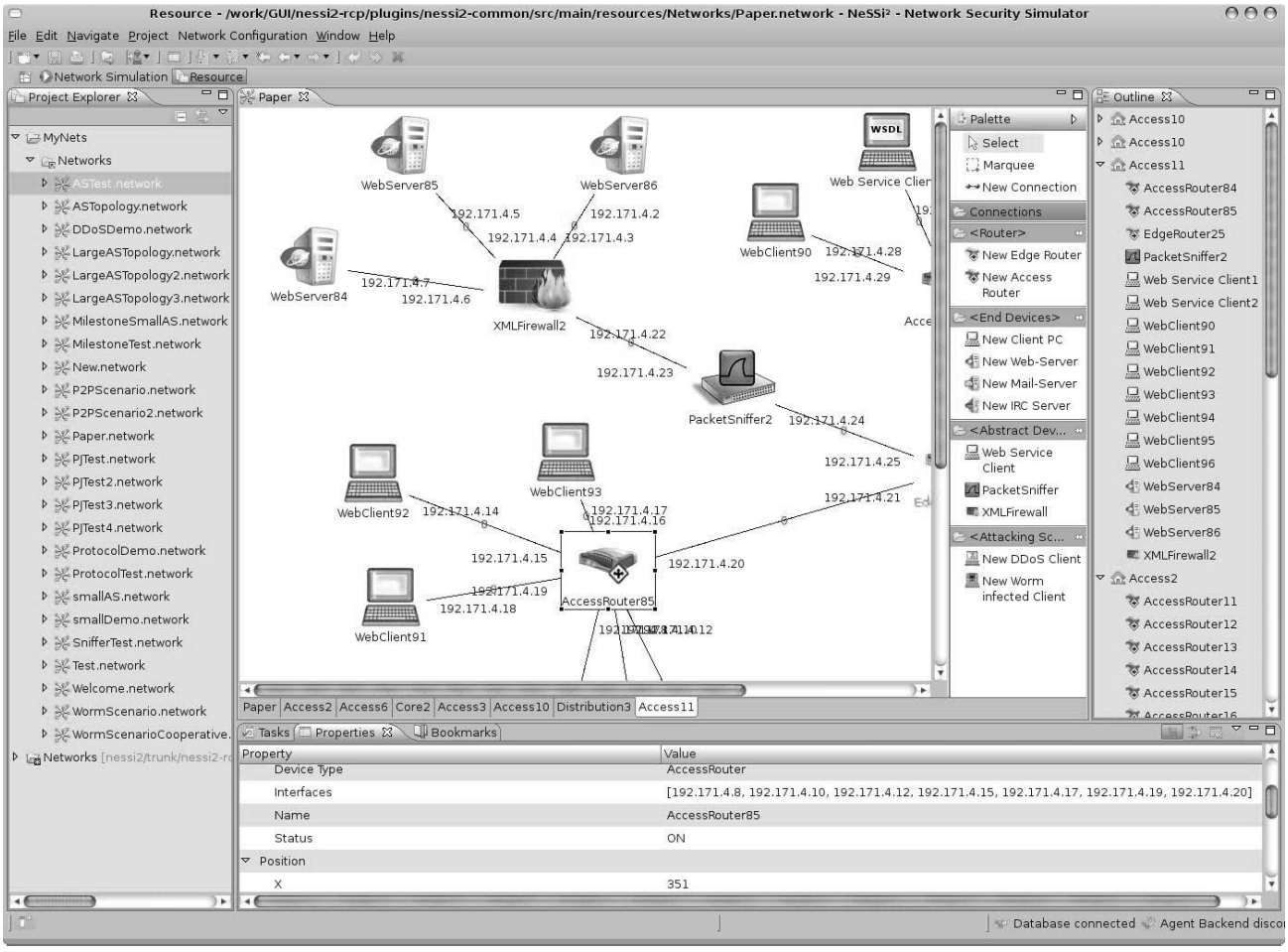

Figure 3. Subnet Editor with user-defined devices such as XML firewalls and packet sniffers

4600 blade server (eight blades, eight cores per blade). Once a session is submitted for execution, the simulation backend parses the desired session parameters (which event types to log, how many runs to execute, etc.), creates a corresponding simulation environment, sets up the database connection and schedules the simulation to run as soon as the necessary processing resources are available.

3.2.1 JIAC Agent Framework

NeSSi is built upon the JIAC framework [13]. JIAC is a service-centric Java-based middleware architecture based on the agent paradigm. Within *NeSSi*, agents are used for modeling and implementing the network entities such as routers, clients and servers. The underlying JIAC agent framework provides a rich and flexible basis for implementing and testing of various security deployments and algorithms in *NeSSi*. Moreover, building upon an agent framework allows the combination of the partial knowledge of the agents residing in the network in a cooperative approach for identifying and eventually eliminating IP-based threats. For example, this may be achieved by monitoring the structure of the encountered IP traffic and the behavior of potentially compromised target systems. For an illustrative example where we successfully utilized the agents' cooperative features, refer to Section 5. Although the software agents provide powerful application-level capabilities, their complexity unfortunately also affects the scalability of the simulation. *NeSSi* mitigates this problem by building upon a parallel-execution model.

3.2.2 Parallel-execution Model

Simulations in large-scale networks are very costly in terms of processing time and memory consumption. Therefore, *NeSSi* has been designed as a distributed simulation, allowing the subdivision of tasks to different computers and processes in a parallel-execution model. *NeSSi* introduces an additional layer of abstraction at the level of network design by explicitly modeling subnets (which

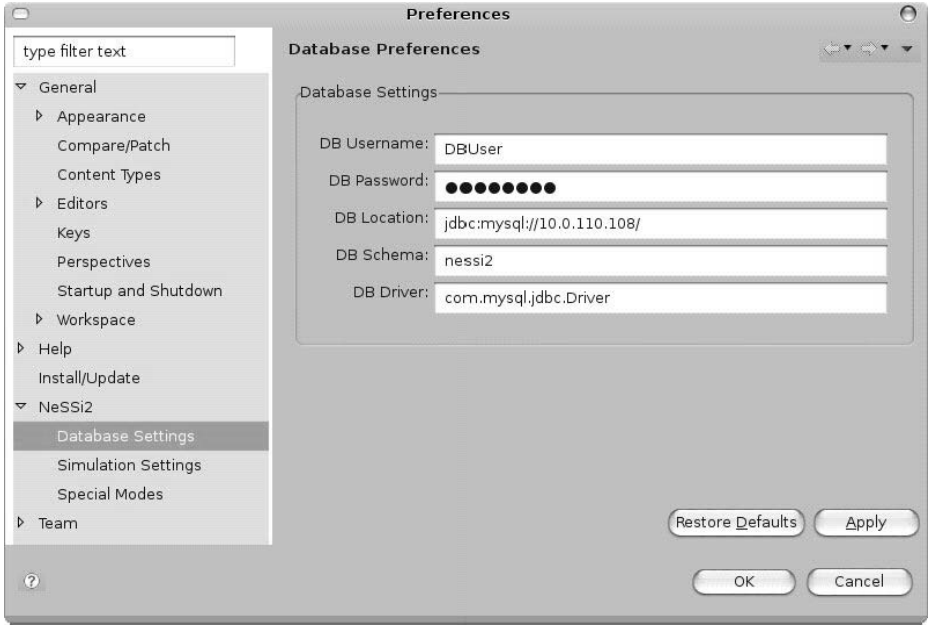

Figure 4. Preference pages allow the user to control all aspects of NeSSi: from local preferences to remote repository locations

constitute the nodes of the abstract network view in Figure 1). This abstract view on the network is shown in the main editor area, which contains core, distribution and access subnets. In this instance, the network depicted in Figure 1) contains three core, three distribution and nine access subnets. In the property sheet below the editor area, the properties of the selected element is visible; in this case it displays that for the *Access1* network, the network address is *130.149.3.0* and the subnet mask is *255.255.255.0*.

Each of these subnets can in turn be opened (via double-clicking or the outline view on the right) to show the devices of the subnet. An example of a subnet open in the main editor window can be seen in Figure 3, where a section of an access subnet containing web clients and servers, firewalls and access routers is visible. This view also contains more detailed information such as network interfaces or link bandwidths. In addition, actual traffic in the runtime phase will be visualized on the links by color and thickness, which allows the selection of 'interesting' links in order to obtain even more detailed statistical and graphical information (cf. Figure 5).

In *NeSSi*, the handling of subnets and the contained devices is distributed to several processes on different individual machines, allowing the simulation to scale better with increasing network size. Figure 6 shows the main components of NeSSi, backend, GUI and *Simulation Database*, as well as their interaction. In the GUI, the network with appropriate scenario and session information is created (cf. Section 3.1.1) and transmitted to the backend, where the simulation is executed. In this regard, log data is stored in the *Simulation Database* and can be accessed again by the GUI to show the results of a simulation run (cf. Section 3.1.2).

In the backend, different agent roles carry out the task of the parallel simulation execution. On the *Master* as well as *Slave Simulation Node*, *Subnet Agents* (SAs) and a *Platform Coordination Agent* (PCA) are running. In addition, the Master Simulation Node executes the *Network Coordination Agent* (NCA).

Subnet Agent (SA). A SA is responsible for managing an individual subnet. During execution, the SA receives events from the PCA (see below) indicating that a new tick has started. It performs all relevant operations such as packet forwarding and application handling that need to be executed for the current tick (an atomic time frame in the discrete-event model). Once all events have been handled, the SA reports back to the PCA that the tick has been successfully executed.

Platform Coordination Agent (PCA). On a single platform, different subnet agents are created, configured and controlled by a PCA. A platform is meant as a software environment for agents running on one computer. The number of SAs handled by the PCA is dependent on the available computing resources. The PCA receives events from the NCA and relays them to all of the SAs residing on its platform. As soon as all SAs have reported back that a tick has been executed successfully, the PCA is responsible

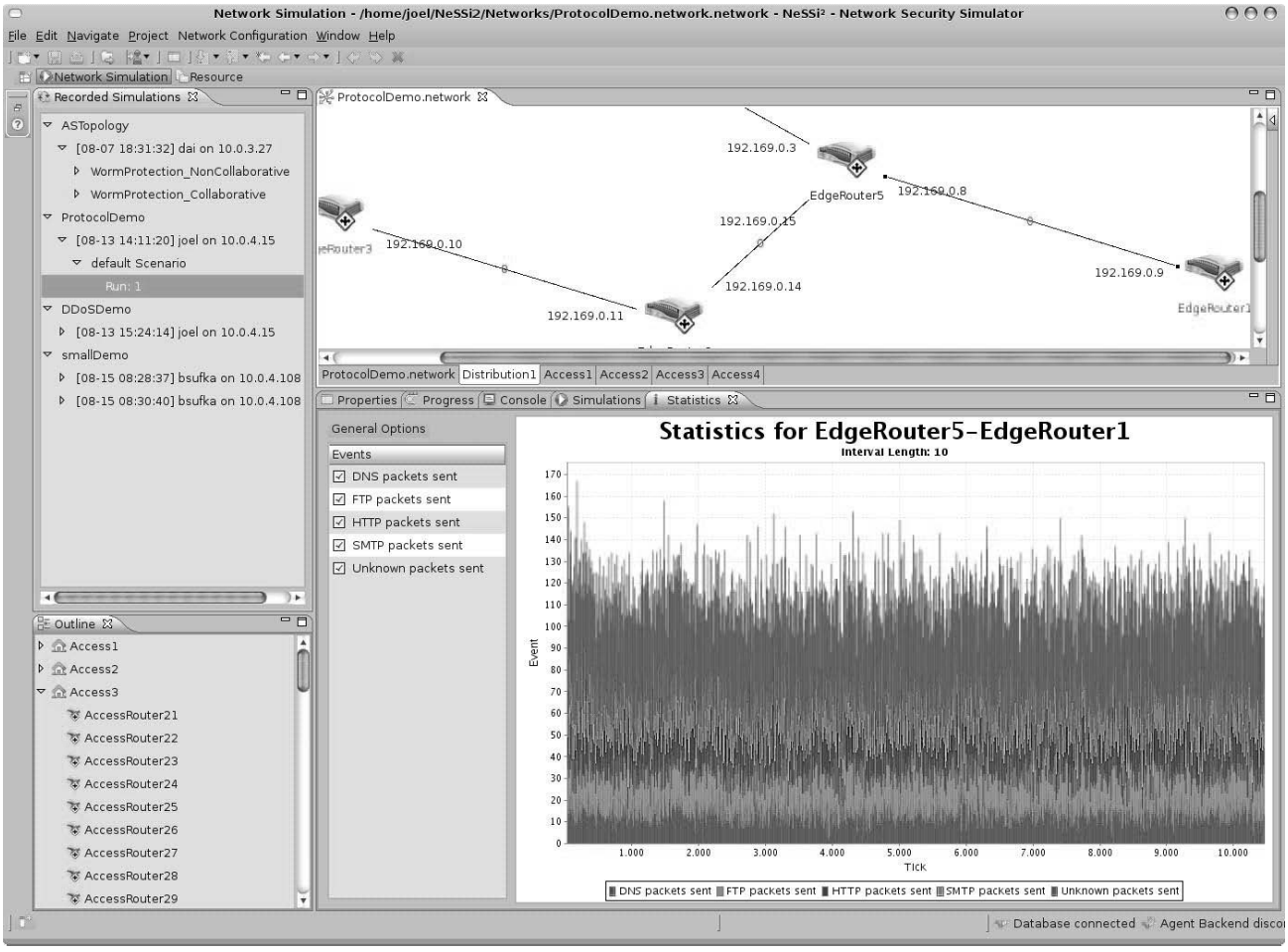

Figure 5. Graphical depiction of traffic in the *Network Simulation* perspective

for forwarding the packets traveling from one subnet to another subnet on a different platform to the respective target PCA. Subsequently, it informs the NCA that the platform has finished the requested task. The platform also sustains a connection to the database, where simulation results and history can be stored (see Section 3.3).

Network Coordination Agent (NCA). The entire network simulation is controlled by the NCA. The NCA coordinates the distribution of the subnets at the beginning of a simulation as well as the synchronization of the platforms. The NCA receives its network model data from the GUI agent. The GUI can also trigger the beginning and determine the end of the simulation, but the NCA does not depend on a GUI agent to be present once it has been started. This architecture allows the distributed network simulation. In addition, the GUI can be disconnected from the simulation environment and later reconnected to it, even from another computer. Figure 6 is a graphical representation of the distributed agent architecture.

3.3 Persistent Session Management

In *NeSSi*, we refer to a scenario where we generate traffic via pre-defined profiles (cf. Section 3.1.1) on a single network over a certain amount of time as a *session*. The accurate reproduction of a session enables users to use different detection methods or various deployments of detection units for the same traffic data set. This allows the comparison of performance and detection efficiency of different security framework setups.

For these purposes, we use a distributed database in which the traffic generated during a session is stored. For each session, the agents log the traffic and detection data and send it to the database that occurs in a simulated scenario between a start and end time. The data types to be logged are specified by the user in the session parameters.

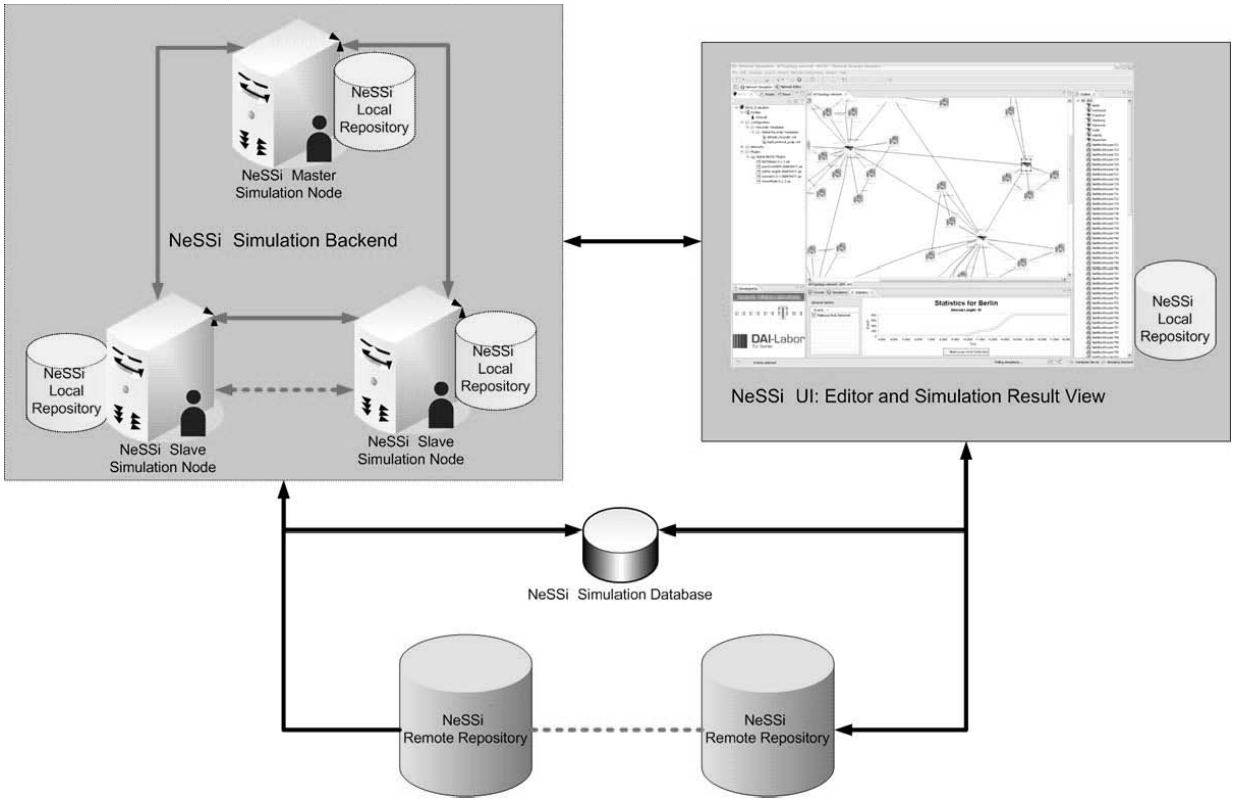

Figure 6. Architecture for distributed simulations with *NeSSi*.

The network model is saved in an XML file. This network file is stored and annotated with a version number based on its hash code in order to link a network uniquely to a session. In addition, attack-related events can be stored in the database for evaluation purposes. Those events are explained in greater detail in Section 5.3.

Owing to the modular design of the database component, the actual implementation of the database is up to the user. This can be comfortably managed via the GUI, where the location of the database, database user name, password and the database driver can be specified. *NeSSi* has been tested extensively with a MySQL database server, but depending on the user's preferences, this can easily be substituted by another module (see Figure 4). The event types to log are specified independent of the underlying implementation via XML-based log4j logger repository settings where multiple appenders can be configured. For example, for simple scenarios the user can simply write the results to a text file if they so desire.

4. The Features of NeSSi

NeSSi is designed to extend conventional network simulation tool features by supporting detailed examination and testing opportunities of security-related network algorithms, detection units and frameworks. The main focus of *NeSSi* is to provide a realistic packet-level simulation environment as a testbed for the development of new detection units as well as existing units. We use *detection unit* as an abstract term for any algorithm or tool employed for the purpose of detecting malicious activity such as intrusion or service degradation attempts.

NeSSi has been designed as a modular application with the focus on extensibility. In particular, *NeSSi* provides out-of-the-box support for various protocols of the TCP/IP stack; a selection of these is described in Section 4.2. These features provide the basis for designing, incorporating and simulating detection units. The framework provided by *NeSSi* for simulating is subsequently described in Section 5.2. The general plug-in API for integrating new user-designed protocols, device types and detection units is beyond the scope of this paper.

4.1 Traffic Generation

Network traffic in the form of IP packets, complete with header and body, can be generated by different means. Implementing the TCP/IP protocol stack, *NeSSi* features an application layer module which is based on standard

Java socket implementations. *NeSSi* incorporates several application-level protocols (HTTP, SMTP, etc.) and supports static and dynamic routing protocols, which can be selected by the user. At the moment, static and dynamic protocols are implemented. A static routing protocol centrally computes the shortest paths as the network is loaded and each time the topology changes. The resulting routing tables are subsequently loaded onto the individual network nodes. On the other hand, IS–IS (intermediate-system-to-intermediate-system) has been implemented as a link-state protocol, which relies on a decentralized algorithm during which routers exchange information about their link states and gather topology information locally. In an iterative fashion, the routing tables at the individual nodes are updated through control message exchange. *NeSSi* can also easily be extended to support a variety of other applications (see Section 4.2).

For this purpose, it is necessary to implement, in addition to the application, the previously described action with the required parameter. In case of a DDoS application, this would be source and destination port, the attacked IP address as well as the packet frequency. Through this, the user's own enhancements can be easily configured, deployed in a network and simulated with *NeSSi*.

4.2 Protocol Stack and Socket-based API

The TCP/IP reference model is the *de-facto* standard for Internet communication. Owing to its importance, *NeSSi* also offers an implementation for it. The routers as well as the end devices in the simulation contain a Network Layer; end devices also exclusively have a Transport and an Application Layer. At the Network Layer, IPv4 is realized with the key features global addressing, routing and fragmentation support. Moreover, TCP/IP model implementation allows containing several protocols in each layer; hence, we also provide IPv6 support in *NeSSi*. For the fault management, TTL (time to live) and header checksums are supported and the ICMP protocol has been implemented for failure notification. Network Layer communication is realized directly on the drop-tail queues that are located at the network interfaces.

On the next level, the Transport Layer is comprised of the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). TCP in *NeSSi* offers a reliable and in-order delivery of data. Sockets represent the interface to the Application Layer, i.e. applications can set up several stream sockets as well as the corresponding server sockets. In this fashion, third-party Java libraries can easily be integrated into the simulation by substituting Java sockets with *NeSSi* sockets. As a proof-of-concept, the JavaMail API[1] has been successfully adapted in *NeSSi*. All applications that are run in *NeSSi* follow a common

1. See http://java.sun.com/products/javamail

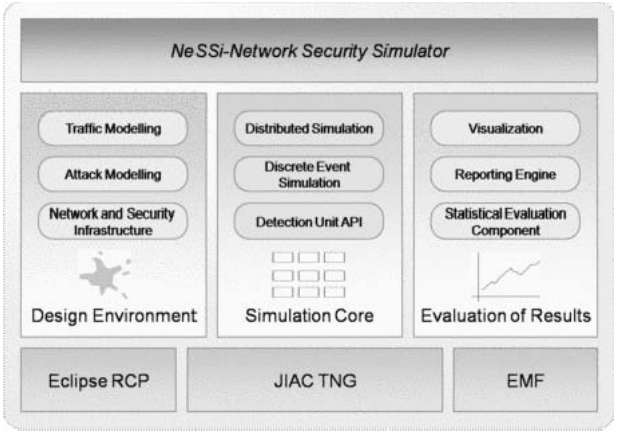

Figure 7. NeSSi comprises several components presented in the middle layer; the bottom layer depicts the main technologies *NeSSi* relies upon.

interface that abstracts from their specific behavior but allows a standardized way of executing them. Currently, the HTTP, SMTP and IRC protocols are integrated in *NeSSi*.

Generated traffic in *NeSSi* can also be exported to files in the *pcap*[2] format in order to inspect the data with standard traffic inspection tools such as wireshark[3]. Several tests have been conducted with the traffic exported from *NeSSi*, verifying that the generated traffic is well formed. Wireshark is able to reconstruct the application data stream without errors if all corresponding IP packets are in the exported file. The generated traffic can be analyzed either offline, for example with the aforementioned tools, or online within the application itself, for example by displaying a graphical summary (cf. Figure 1).

5. Security Simulation Features

In the previous sections, we described the basic network model and traffic-generation features of *NeSSi* as well as other aspects regarding general network simulation. The distinguishing feature of *NeSSi* is the focus on network security framework and algorithm evaluation. Figure 7 provides a conceptual view on *NeSSi*. The bottom layer shows some of the important technologies that *NeSSi* uses. This includes the JIAC agent framework used for representation of the different entities in the simulation, EMF for the network data model and Eclipse RCP as the platform-independent, plug-in-based execution environment. On top of the bottom layer, several concepts realized in *NeSSi* can be found. These are the aforementioned general network simulation components such as traffic modeling via

2. See http://www.tcpdump.org
3. See http://www.wireshark.org

node profiles, a parallel execution model, discrete-event simulation and the visualization.

In this section, we presume that a network topology and background traffic have been created, as described in the previous sections. We now highlight the security simulation capabilities of *NeSSi* that can be added to this basic setup in a four-step process. First, we describe how attacks are modeled and generated in Section 5.1. Subsequently, detection capabilities are added to the network in order to evaluate their efficiency in detecting the configured attack scenarios (Section 5.2). Finally, Section 5.3 demonstrates how the results of the simulation are presented to the user in order to draw conclusions as to whether the selected detection approach is performing well in protecting against the configured attack.

In a final step, we show how the abovementioned concepts are put into practice by means of an example scenario. Using the example of detecting and containing worm spread in a network of multiple autonomous systems, Section 5.4 applies the concepts from the Sections 5.1, 5.2 and 5.3 in a step-by-step process.

5.1 Generating Attacks

The simulation setup in *NeSSi* not only comprises network creation and attachment of traffic profiles (cf. Section 3.1.1), but additionally security related settings can be configured. When a security framework composed of several detection units is to be tested, profiles can also be used in *NeSSi* to simulate attacker behavior and attack patterns. Accordingly, *NeSSi* provides out-of-the box support for various attack scenarios such as bot networks initiating DDoS attacks. Here, infected end device nodes, 'zombies', are controlled by the bot net commander via the Internet Relay Chat application. The commander is capable of initiating different kinds of DDoS attacks such as the SYN Flooding or UDP Storm. To this end, the attacker connects to an IRC communication server and sends attack commands to a chat channel that all of the bots are listening to. As a result, the bots execute the desired attack.

In addition, worm propagation schemes are supported. Here, the behavior of the SQL Slammer worm and the Blaster worm have been realized exemplarily in *NeSSi*. In general, a worm propagation scheme as well as the susceptible–infected–recovered (SIR) model as an epidemiological model is included in *NeSSi*. The worm propagations scheme can be configured in different ways: on the one hand, there exist configuration files defining the number of initial propagators and where in the network the outbreak should take place. In addition, the spreading scheme, i.e. which IP addresses to attack and delays in between, can be set.

On the other hand, the worm application model itself can either be extended or a new model can seamlessly be integrated via the application interface provided in *NeSSi*

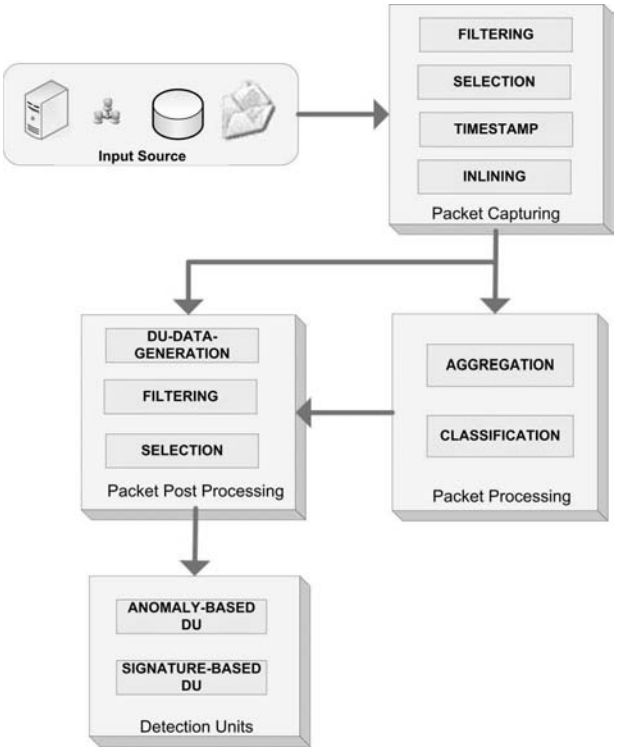

Figure 8. The components of the Detection Unit API and the packet/data flow between them; the Packet Processing component is not necessary for all detection algorithms

(cf. Section 4.2). In the second step, newly developed or pre-configured detection units can be deployed on a set of links in the simulation. Therefore, analogous to the traffic profiles, detection profiles can be created. Those detection profiles consist of one or more detection units and can be deployed on end devices, links or routers. Details of how these deployments are evaluated with regard to their efficiency can be found in Section 6.2. The tools used for these evaluation tasks are described in the next section.

5.2 Detection Unit API

First and foremost, *NeSSi* provides a *Detection Unit API* for the development of new detection algorithms as well as the integration of existing algorithms. The architecture for this purpose is shown in Figure 8. It consists of four main components whose activation depends on the configuration of the detection approach; the *Packet Capturing* component is mandatory and processes incoming traffic from a data source. This is usually a link in the *NeSSi* simulation but can also be file system streams or a database connection. Here, packets are selected according to filter rules and a sampling policy (such as 'every tenth packet') to narrow down the processing over-

head. Accordingly, the packets are associated with time stamps.

Several detection algorithms, e.g. behavior-based approaches, do not only process packets but also related statistical information. As an example, the well-known *SYN Flood* attack is characterized by a massive amount of open TCP connections. In this case, the *Packet Processing* component offers the construction of IPFIX[4] data flows based on the packet data. The flow specification is open to the developer by configuration files. To this end, a tree structure of the flow is defined by providing key attributes and optional data fields. A data flow representing distinct TCP sessions must have at least a source IP address, destination IP address, source port, destination port and a protocol flag as key attributes.

Moreover, the *Packet Post Processing* component generates the actual input to the detection units. This input can also be specified by a detection unit which ranges from raw packets for signature-based schemes such as virus scanners, to complex ratios of traffic statistics based on the flow data; for example, in the case of a SYN Flood attack, the ratio of half-open TCP connections to all TCP connections can be specified.

Finally, the *detection units* may be well-known security solutions as contemporary commercial virus scanner software or new tools developed in scientific research projects. In *NeSSi*, both can be incorporated as long as they adhere to a specified interface. The configuration of a detection unit and the required components are stored in a *template*. Furthermore, additional properties such as *Inlining*, i.e. synchronous packet processing, can be set. This allows the application of counter measures. A processing interval for a detection algorithm is another option that can be set here.

5.3 Reporting and Evaluation

NeSSi allows the simulation of various security scenarios. In addition, there is a huge diversity in network security evaluation metrics. Here, the developer of a detection algorithm respectively of a special security infrastructure set-up may not only be interested in detection rates, but also in the economical assessment of a scenario. Hence, the gathering of simulation results and the evaluation needs to be very flexible. Here, we apply an event-based approach, the so-called *Meta Attack Events*. Already included events incorporate dropped packets, infected flows, compromised machines, unavailable services, etc. Those events are stored in the database at runtime. Events belonging to the same attack refer to a global ID to differentiate between the impacts of different attacks. The database associates those events with a time stamp in the simulation as well as a device and/or transmitted packets related to that specific event.

4. See http://www.ietf.org/html.charters/ipfix-charter.html

Furthermore, we apply BIRT (Business Intelligence and Reporting Tools) [14] for visualization and analysis of results. BIRT comprises, on the one hand, a graphical report designer capable of creating report templates. In those templates, different input sources such as databases, simple Java objects or XML files can be declared. In addition, BIRT allows standard statistical operations on the data and enables the choice of several chart types to display the resulting data series. More complex preprocessing can easily be carried out by adding Java or JavaScript code.

On the other hand, BIRT offers an environment for the generation of the reports at runtime. In this case, a report template is loaded and the actual selected data source, in the example of *NeSSi* usually a simulation session is bound to the report. Subsequently, an HTML document containing the desired report is generated and can be shown in any Internet Browser. Figure 9 shows an example report created in the *NeSSi* environment. It compares detection rates of different algorithms; the detection rate is measured as detected number of infections in comparison to the total number of infections (sensitivity).

5.4 Example Scenario: Signature-based Worm Spread Detection and Prevention

Computer worms represent a serious threat to communication networks, even motivating the development of several special-purpose simulation tools to examine their properties (cf. Section 2). Therefore, we analyzed how the impact of an epidemic that spreads too fast to be contained using human supervision processes can be minimized with signature-based countermeasures. To be able to model realistic scenarios, we extended *NeSSi* to support multi-autonomous system topologies. In the following, we show how this simulation study is conducted; selected steps are shown in Figure 10.

In the beginning, the AS topology is created using the wizard for parameter-based automated network creation. An extract is shown in the main editor window in the background of Figure 10 (step 1). Next, this scenario requires a worm and a detection unit application. As shown in Figure 10 (step 2), applications can be customized by specifying various parameters. In this manner, it is possible to emulate the behavior of existing malware such as *CodeRed* or *SQL Slammer*. We used a CodeRed-like scanning behavior with small adaptations due to the smaller IP space. The worm application has two states: susceptible and infected. Initially, only a few worm applications are in the infected state, but they will be switched if they receive a worm packet on the vulnerable port. The detection application utilizes network error messages (ICMP). If hosts cause too much ICMP traffic, they are marked as suspicious and the detection application starts to generate signatures for their traffic. After a predefined amount of a certain signature has been created for different hosts,

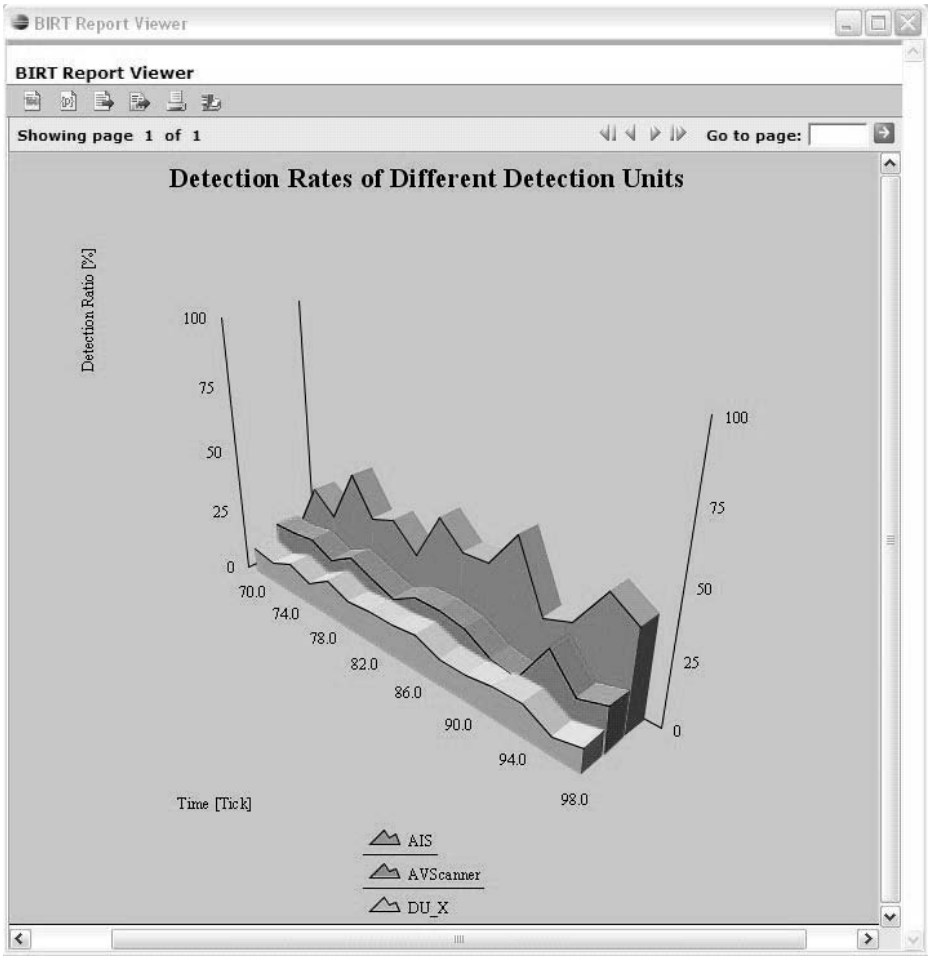

Figure 9. Reporting in NeSSi: this example report shows an evaluation of the detection ratio (sensitivity) of different detection algorithms

it will be deployed and prevents infections. For multi-autonomous system topologies, the applications are deployed automatically. Alternatively, manual configuration as shown in step 3 is also possible. Now that both the attack and the detection profiles have been set up, the user has to create, configure and execute a session.

Finally, the simulation can be evaluated in the corresponding perspective. The results of this example run are pictured in Figure 1. Here, we differ between the spread inside and outside the protected autonomous system. For the latter, the events are split into two graphs, the infections inside the protected autonomous systems and the averted attacks after the signatures where deployed. Here the mitigation was successful as less then a quarter of the vulnerable hosts have been infected.

In the following section, we discuss further security research conducted with *NeSSi* to demonstrate that it is not merely a worm simulation tool but has been successfully used to conduct a wide variety of simulation studies in various fields.

6. Recent and Published Research Activities

NeSSi has already demonstrated its value in recent research and was employed as a test framework for various network-centric security approaches.

6.1 Artificial Immune System

As an example, a distributed and collaborative *Artificial Immune System* (AIS) was implemented in *NeSSi* for testing an anomaly-based detection algorithm [12]. In this scenario, anomaly detection units on different hosts compute the probability of an anomaly by an AIS component. The idea of AIS is to compute the possibility of an anomaly by comparing the actual status of the system to a set of detectors created by negative selection [15]. This algorithm is a metaphor of the biological negative selection taking place during the maturation of the immune cells. The detectors are produced nearly randomly and compared with the normal data; if a detector is similar to a

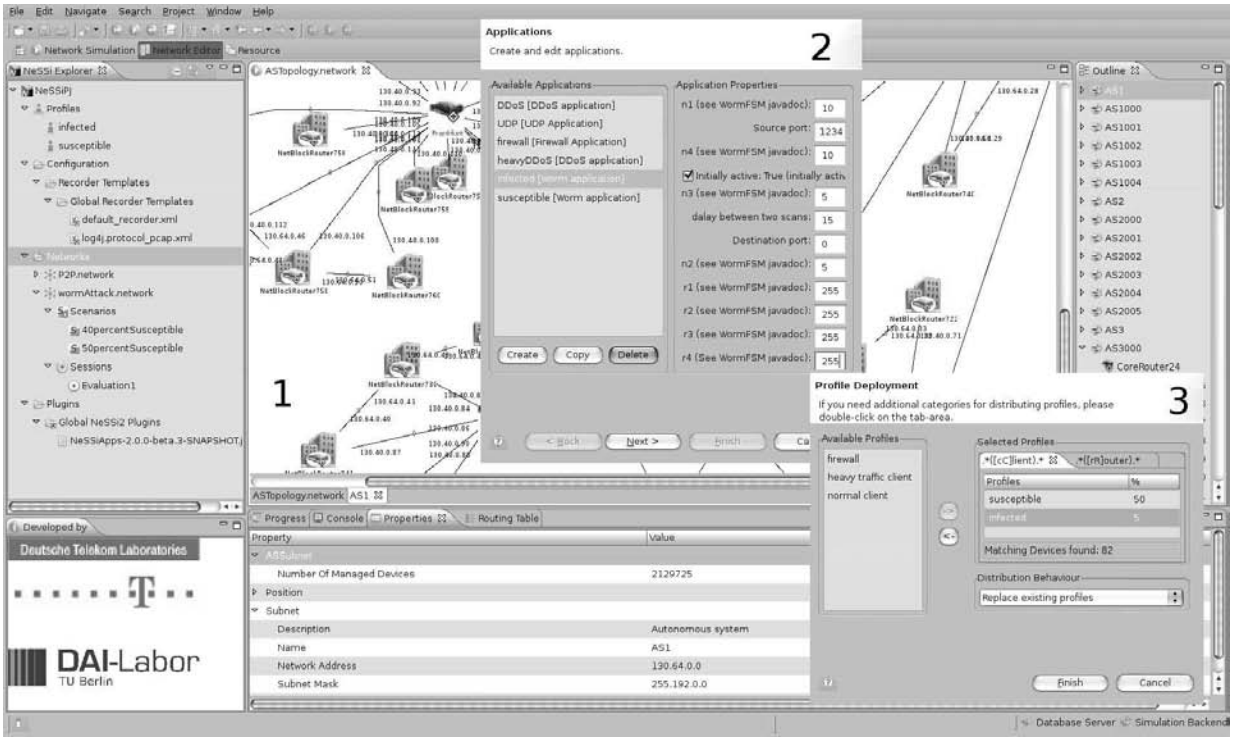

Figure 10. Setting up a security simulation: worm spread inside a topology of multiple autonomous systems

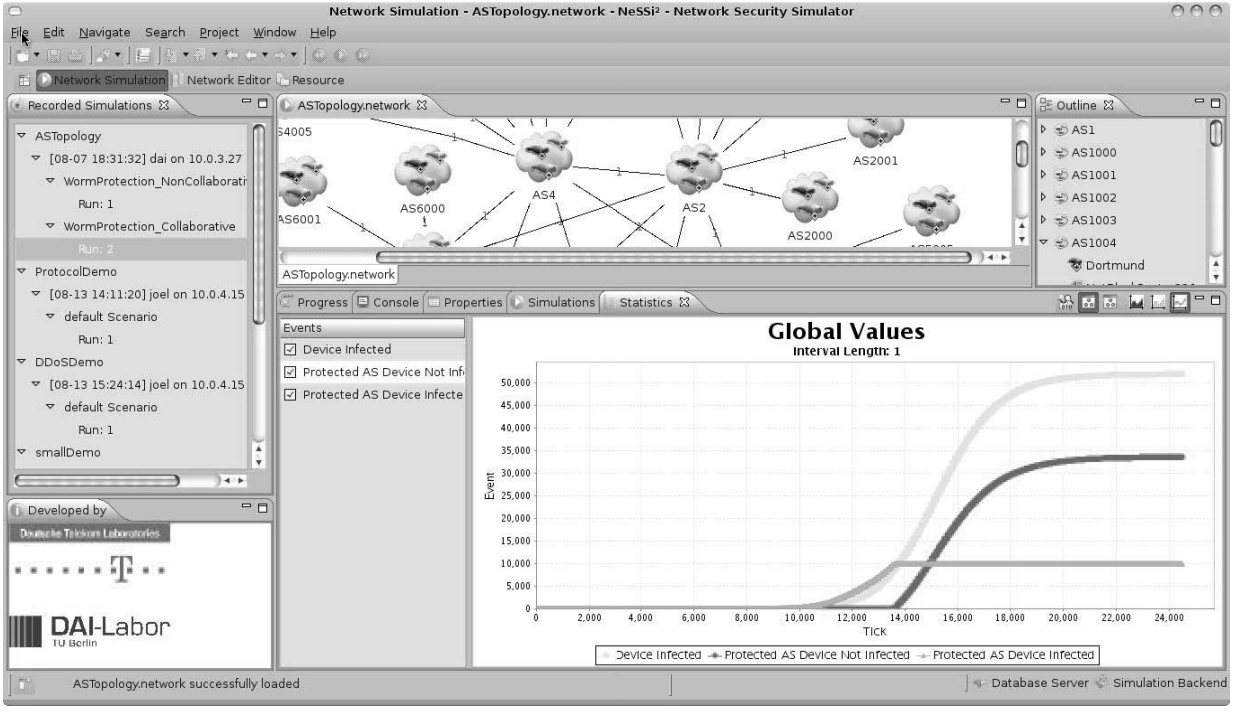

Figure 11. Evaluating worm spread prevention efficiency: spread in protected (lower, blue) and unprotected (upper, yellow) autonomous systems

normal feature vector, it is deleted and a new one is created, and this is repeated until no detector is similar to the normal feature vector set (training set).

The data processed by the AIS is statistical in nature and obtained by a monitoring component on the host agent. Here, the probability of an anomaly constitutes the status of a client. The cooperation between the clients takes place by sharing these status levels and computing new status level information based on the incoming status of others and its own status. The tests results, verified in *NeSSi*, indicated a very good performance of the AIS in general while the false positive rate (the main problem of anomaly detection systems) was lowered significantly. The communication between the AIS clients described in [12] has been implemented in *NeSSi* via a customized peer-to-peer protocol to avoid the single point-of-failure of a central server. This peer-to-peer protocol allows the combination of AISs into detection groups.

As an extension, a scheme for the decentralized detector set generation was presented by Bye et al. [16]. Here, the overall feature space is partitioned in several subspaces administrated by AIS-enabled nodes. As a result, the individual computational load for each node is decreased. Nevertheless, by the application of combinatorial design techniques such as Symmetric Balanced Incomplete Block Design or Generalized Quadrangles, a controlled level of overlap between the detector sets is realized and sets are exchanged deterministically between peers. Finally, this results in a trade-off between the aforementioned computational task on the one hand and a guaranteed level of redundancy on the other hand.

6.2 Detection Device Placement

As a second use-case scenario, we studied various detection device placement approaches. To this end, we used a number of identical detection units in order to ensure comparability. The type of detection unit to use can be selected prior to starting the device placement scenario in *NeSSi*. Furthermore, the configuration dialog allows the selection of how many total detection devices may be placed, representing a total sampling budget. The objective is then to maximize the detection efficiency subject to the budget constraints. Several algorithms are supported for computing sensible deployment strategies, for example Betweenness Centrality [17], Traffic Betweenness Centrality [18] or a selection of game-theoretic approaches [19, 20].

In *NeSSi*, we have already successfully evaluated two of these approaches, namely node selection algorithms using the Betweenness Centrality paradigm as well as a monitor placement game described in [20]. The Betweenness Centrality algorithm originates from social network theory and indicates the importance of a node in the overall network communication. It is run asynchronously in *NeSSi* after a change to the network topology occurs and returns a numerical value for every node indicating its centrality index.

Subsequently, detection devices are placed on the devices corresponding to the nodes with the largest values as an optimal deployment. We then ran a test with default traffic-generation profiles on all clients for a fixed number of execution cycles. For comparison, we subsequently fed the same traffic in the same network but deployed the detection units randomly. The results indicate the superiority of deploying on nodes with high Betweenness Centrality index. We assume that in case traffic is not uniformly distributed in the network, i.e. there are clients who generate considerably more traffic than others, the Traffic Betweenness Centrality [18] approach would be even more suitable. Puzis et al. [21] mention the possibility of adapting the Betweenness Centrality to find a most prominent group in a network, which they define as a set of nodes which covers the most distinct source–destination (client–server) paths.

In a more recent paper, Brandes also considers various variants of Shortest-Path Betweenness Centrality [22]. Some of these are particularly suited for application to real-world scenarios, such as the *proxies* and *endpoints* subproblems. In *NeSSi*, we have applied the algorithms for these variants in order to generate more realistic Betweenness Centrality values.

These applications result in being able to generate *subnet-weighted* Betweenness Centrality values, where it is assumed that traffic in subnets will be proportionally larger than intra-subnet traffic. A second variant takes into account that only end devices will initiate connections and generate traffic while the intermediate routers merely forward it. This enables us to disregard all shortest paths (s,t) where s or t is a router, leading to more meaningful centrality values.

In the second study, a game-theoretic approached was employed where the attackers and the IDS are modeled as adversaries in a two-person game. We were able to demonstrate the existence of a mixed-strategy saddle-point equilibrium and computed its value for two example networks. The adversaries are modeled as *Runnables* in *NeSSi*. Upon starting the simulation, the respective threads are started concurrently, and elementary player actions from the strategy matrix are selected using a random number generator. It was possible to verify the game-theoretic optimality and measure the performance of the obtained strategies to the devised network security game.

Finally, hierarchical intrusion detection approaches are emerging as important tools in managing the increasing complexity of network-based intrusion detection systems. This is particularly true for *ad-hoc* networks, where the lack of central audit points mandates a distributed and adaptive intrusion detection approach [23]. The hierarchy needs to be constantly able to adapt to nodes leaving and entering the network. In this respect, we have implemented Basagni's algorithm [24] for performing a distributed clustering and determining the clusterheads of the intrusion detection hierarchy. This is a first step towards building a multi-level intrusion detection hierarchy as de-

scribed in [23]. We expect to obtain relevant results as soon as *ad-hoc* network protocol support in *NeSSi* through plug-ins reaches a mature level.

6.3 Collaborative Intrusion and Malware Detection

Another case study realized in *NeSSi* took place in the scope of CIMD (Collaborative Intrusion and Malware Detection). The CIMD framework enables cooperative intrusion detection approaches via an overlay network. For this purpose *detection groups*, i.e. groups with a common *objective* for intrusion detection are built. Each participant of CIMD is characterized by attributes, e.g. hardware, software or network configuration. Based on this data model, the *interests*, i.e. characteristics of desired group members can be expressed. Subsequently, the overlay network performs the look-up of potential group members and conducts the grouping. Bye et al. introduced the main aspects of CIMD which are the data model, grouping algorithm and matching function(s) in [25]. In this work, also vulnerability analysis and realization strategies of the system are given as well as simulation and analysis was conducted pinpointing the benefit of the overall approach.

In this regard, the case study of *heterogeneous detection groups* for signature exchange was applied. These groups of disparate NIDSs follow the common objective of 'signature exchange' while originating from distinct manufacturers. Customer networks are connected to a NSP and provided a portfolio of disparate NIDS products as a commercial service. Each customer network is protected by a randomly selected NIDS type installed at the gateway to the NSP. The attack vector bases on *drive-by downloads*, i.e. malicious web servers exploit browser vulnerabilities of a user when accessing web sites. In this regard, clients in the customer networks continuously access the web servers and become infected in the case the server is malicious. However, in the case that a signature for the attack is already available on the NIDS, the attack is prevented. In the beginning, attacks are unknown and devices are exposed to the vulnerability window, i.e. time between an exploit becomes available and appropriate signature is generated. However, signatures for the threats become available with different update times depending on the manufacturer. In the non-collaborative scenario, the NIDS are working on their own, whereas in the collaborative scenario the heterogeneous detection groups are built in advance with the purpose to mediate upcoming signatures between the distinct participants.

The benefit of the detection groups has been evaluated in *NeSSi*. Here, a topology of 58 access networks connected to the backbone of the NSP was realized (cf. Figure 12). The NIDS portfolio was composed of three different products, which were equally distributed over the customer networks. Signatures for the attack become available in the interval between a minimum and a maximum update time. We simulated the collaborative and non-collaborative in *NeSSi* with varying malicious web server probabilities. The results of the simulation showed a reduction of the number of infections compared with the non-collaborative approach. Probabilistic analysis was also conducted to support the simulation results showing also that collaboration becomes the more feasible the more heterogeneous the detection groups are.

6.4 Detection Algorithm Evaluation

In a university graduate class, *NeSSi* was used to produce data for the evaluation of detection algorithms implemented by students without any knowledge of the simulator. When it is not required to analyze simulation data at runtime, *NeSSi* allows the storage of simulation data, e.g. the IP packets, in the database. In the anomaly detection class, *NeSSi* was used to simulate non-infected traffic, a DDoS attack and a worm attack. The students were tasked with implementing an anomaly detection algorithm and comparing detection results between different algorithms, attack types and detector placements within in *NeSSi*. All of these different experiments were based on the simulation data stored in the *NeSSi* database. The evaluated detection approaches included a distributed AIS [26], a self-organizing map [27] and naive Bayes classifiers [28].

7. Conclusion and Outlook

We have presented and described the network security simulator *NeSSi*. Based on agent technology and discrete-event simulation, it is a highly scalable, platform-independent network simulation tool with special features for evaluating security solutions. In particular, the plug-in-based API allows security experts to write their own detection unit plug-ins and test them in *NeSSi*.

Network simulation is a very wide and complex area in software engineering. There is an abundance of features existing in real-life networks; moreover, standards are changing and new standards are introduced continuously at a rapid rate. Hence, keeping *NeSSi* up-to-date with the latest trends in network technology is a great challenge. The following is a list of features we plan to focus on in the further development of *NeSSi*. For comparison, a list of features for the next generation of *ns-2* (some of them are already supported in *NeSSi*) can be found in [29].

Usability. In order to appeal to a greater audience, the usability of the simulator can be improved. This includes role-based user-interface perspectives dependent on experience level (novice or expert).

Scenarios. More default attack scenarios will be supported. We will extend the already realized bot net infrastructure capable of executing DDoS-attacks (cf. Figure 10) by peer-to-peer principles. Peer-to-peer bot nets are regarded as an emerging important threat in network security.

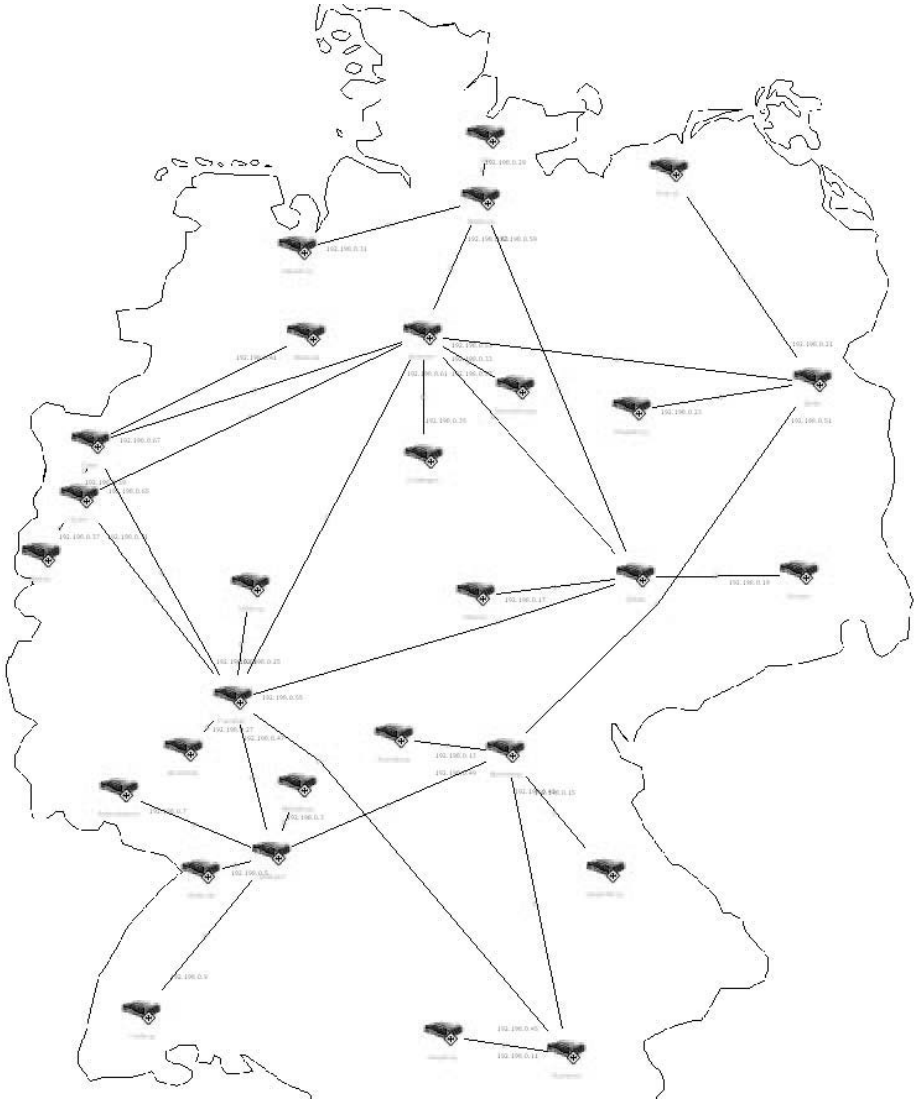

Figure 12. Backbone of the simulated network used for the heterogeneous detection group scenario

Critical infrastructures. In future versions, we aim to extend the strictly IP-based topology models by adding features for examining higher-level security risks, for example in critical infrastructures. In particular, we focus on providing more generic network models which interoperate with standard graph libraries, for example *JUNG* [30]. This will allow the user to examine interdependencies between different types of commodities (i.e. power and communication infrastructures) and assess the related security risks.

Since the end of 2008, *NeSSi* is available under an open-source license. Please visit http://www.nessi2.de for the newest developments.

8. Acknowledgements

The authors would especially like to thank their scientific advisors Tansu Alpcan and Katja Luther for their invaluable support and their fellow colleagues Thorsten Rimkus, Sebastian Linkiewicz, Marcus Lagemann, Thomas Hauschild, Jakob Strafer and Dennis Grunewald for their dedicated work on this project. This research was supported and funded by Deutsche Telekom AG.

9. References

[1] USC Information Sciences Institute. NS-2 network simulator 2.31. http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf (accessed 07/15/2008.

[2] Nicol, D.M., M. Liljenstam and J. Liu. 2005. Advanced concepts in large-scale network simulation. In M.E. Kuhl, N.M. Steiger, F. Armstrong and J.A. Joines, Eds, *37th Winter Simulation Conference*, ACM Press, New York, pp. 153–166.

[3] Liu, B., D.R. Figueiredo, Y. Guo, J. Kurose and D. Towsley. 2001. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 3, pp. 1244–1253.

[4] Scalable Network Technologies Inc. Qualnet. http://www.scalable-networks.com.

[5] Riley, G.F. 2003. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, ACM Press, New York, pp. 5–12

[6] Wei, S., J. Mirkovic and M. Swany. 2005. Distributed worm simulation with a realistic internet model. In *PADS 2005. Workshop on Principles of Advanced and Distributed Simulation*, June 2005, pp. 71–79.

[7] Liljenstam, M., J. Liu, D.M. Nicol, Y. Yuan, G. Yan and C. Grier. 2006. RINSE: The Real-Time Immersive Network Simulation Environment for Network Security Exercises (Extended Version). *Simulation*, 82(1): 43–59.

[8] Yun, J.B., E.K. Park, E.G. Im and H.P. In. 2005. A scalable, ordered scenario-based network security simulator. In *Systems Modeling and Simulation: Theory and Applications* (*Lecture Notes in Computer Science,* Vol. 3389), Springer, Berlin, pp. 487–494.

[9] Eclipse Foundation. Eclipsepedia: RCP. http://wiki.eclipse.org/RCP (accessed 10/11/2008).

[10] Eclipse Foundation. Eclipsepedia: SWT. http://www.eclipse.org/swt (accessed 10/11/2008).

[11] Eclipse Foundation. Eclipse modeling framework project. http://www.eclipse.org/modeling/emf (accessed 10/11/2008).

[12] Luther, K., R. Bye, T. Alpcan, S. Albayrak and A. Müller. 2007. A cooperative AIS framework for intrusion detection. In *Proceedings of the IEEE International Conference on Communications (ICC 2007)*.

[13] Fricke, S., K. Bsufka, J. Keiser, T. Schmidt, R. Sesseler and S. Albayrak. 2001. Agent-based telematic services and telecom applications. *Communications of the ACM*, 44(4): 43–48.

[14] Eclipse Foundation. Business intelligence and reporting tools. http://www.eclipse.org/birt/phoenix (accessed 10/11/2008).

[15] Forrest, S., A.S. Perelson, L. Allen and R. Cherukuri. 1994. Self-nonself discrimination in a computer. In *Proceedings of the IEEE Symposium on Research in Security and Privacy,* IEEE Computer Society Press, Los Alamitos, CA, pp. 202–212.

[16] Bye, R., K. Luther, S.A. Camtepe, T. Alpcan, S. Albayrak and B. Yener. 2008. Decentralized detector generation in cooperative intrusion detection systems. In S. Masuzawa and T. Tixeuil, Eds, *Stabilization, Safety, and Security of Distributed Systems 9th International Symposium, SSS 2007 Paris, France, November 14–16, 2007 Proceedings* (*Lecture Notes in Computer Science*, Vol. 4838), Springer, Berlin.

[17] Brandes, U. 2001. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2): 163–177.

[18] Bloem, M., T. Alpean, S. Schmidt and T. Basar. 2007. Malware filtering for network security using weighted optimality measures. In *IEEE International Conference on Control Applications, 2007. CCA 2007*, pp. 295–300.

[19] Kodialam, M. and T. Lakshman. 2003. Detecting network intrusions via sampling: a game theoretic approach. In *Proceedings IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, pp. 1880–1889.

[20] Schmidt, S., T. Alpcan, S. Albayrak and A. Müller. 2007. A monitor placement game for intrusion detection. In *Proceedings of CRITIS, 2nd International Workshop on Critical Information Infrastructures Security* (*Lecture Notes in Computer Science*), 5141, Springer, Berlin.

[21] Puzis, R., Y. Elovici and S. Dolev. 2007. Fast algorithm for successive computation of group betweenness centrality. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 76(5): 056709.

[22] Brandes, U. 2008. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2): 136–145.

[23] Sterne, D., P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng and T. Bowen. 2005. A general cooperative intrusion detection architecture for manets. In *Proceedings of the Third IEEE International Workshop on Information Assurance*, pp. 57–70.

[24] Basagni, S. 1999. Distributed clustering for ad hoc networks. In *Parallel Architectures, Algorithms, and Networks, 1999. (I-SPAN '99) Proceedings. Fourth International Symposium on*, pp. 310–315.

[25] Bye, R. and S. Albayrak. *CIMD—-Collaborative Intrusion and Malware Detection*, Technical Report TUB-DAI 08/08-01, Technische Universität Berlin, DAI-Labor, August 2008, http://www.dai-labor.de.

[26] Hofmeyr, S. and S. Forrest. 2000. Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4): 443–473.

[27] Höglund, A.J., K. Hätönen and A.S. Sorvari. 2000. A computer host-based user anomaly detection system using the self-organizing map. In *IJCNN '00: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00),* Vol. 5, IEEE Computer Society, Washington, DC, p. 5411.

[28] Amor, N.B., S. Benferhat and Z. Elouedi. 2004. Naive Bayes vs decision trees in intrusion detection systems. In *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*, ACM Press, New York, pp. 420–424.

[29] ns-3 project. NS-3 network simulator. http://www.nsnam.org/docs/architecture.pdf (accessed 07/15/2008).

[30] JUNG. 2003. Java universal network/graph framework software library. http://jung.sourceforge.net (accessed 26/10/2008).

Stephan Schmidt *received the Diplom-Informatiker (comparable with international master's degree) in 2005 at Technische Universität Berlin. Currently he is working as a researcher and project leader at DAI-Labor at Technische Universität Berlin and pursuing his PhD.*

Rainer Bye *received the Diplom-Informatiker in 2005 at Technische Universität Braunschweig. Currently he is working as a researcher and project leader at DAI-Labor at Technische Universität Berlin and pursuing his PhD.*

Joel Chinnow *received the Diplom-Informatiker in 2009 at Technische Universität Berlin. Currently he is working at DAI-Labor Technische Universität Berlin and pursuing his PhD.*

Karsten Bsufka *received his degree in 2006 from the Technische Universität Berlin. He currently works has a senior researcher at the DAI-Labor and coordinates research projects related to network security and network simulation and teaches classes on network simulation and intrusion detection. He was the lead developer for the security mechanisms in the JIAC IV agent framework and responsible for the Common Criteria evaluation of JIAC IV.*

Seyit Ahmet Camtepe *received a PhD (2007) in Computer Science from Rensselaer Polytechnic Institute, Troy, NY, and BSc (1996) and MSc (2002) degrees in Computer Engineering from Bogazici University, Istanbul, Turkey. He worked as a Network and Security Engineer in Pamukbank (currently Halkbank), Istanbul, Turkey, from 1996 to 2002. He is currently pursuing his habilitation and directing the Competence Center Security in DAI-Labor at Technische Universität Berlin.*

Sahin Albayrak *is the chair of the professorship on Agent Technologies in Business Applications and Telecommunication (AOT) at Technische Universität Berlin. He is the founder and head of DAI-Labor, currently employing about 100 researchers and support staff. He is a member of IEEE, ACM, Gesellschaft für Informatik (German Computer Science Society, GI), and AAAI. He is one of the founding members of Deutsche Telekom Laboratories (T-Labs) and currently a member of its steering board.*