Application-level simulation for network security

Rainer Bye, Stephan Schmidt
Katja Luther and Sahin Albayrak[1]
DAI-Labor, Technische Universität Berlin
{rainer.bye, stephan.schmidt, katja.luther, sahin.albayrak}@dai-labor.de

ABSTRACT

We introduce and describe a novel network simulation tool called *NeSSi* (**Ne**twork **S**ecurity **S**imulator). *NeSSi* 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 interface support for the plug-in of detection algorithms allow it to be used for security research and evaluation purposes. *NeSSi* has been utilized for testing intrusion detection algorithms, conducting network security analysis, and developing distributed security frameworks at the application level. *NeSSi* is built upon the agent componentware framework *JIAC* [5], resulting in a distributed and easy-to-extend architecture. In this paper, we provide an overview of the *NeSSi* architecture and briefly demonstrate its usage in three example security research projects. These projects comprise of evaluation of stand-alone detection unit performance, detection device deployment at central nodes in the network and comparison of different detection algorithms.

Categories and Subject Descriptors

I.6.3 [**Simulation and modeling**]: Applications

General Terms

Security, Design

Keywords

Network simulation, network security

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 travelling 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 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 realistic 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 with focus on security evaluation capabilities. We then describe the general software architecture of *NeSSi* in Section 3, focus on the security features in Section 4 and subsequently demonstrate how it can be used for the setup and execution of realistic attack scenarios. Finally, an outlook on future extensions of *NeSSi* is given in Section 6.

2. CONTRIBUTION

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* [15]. *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 [10] and [8]. Discrete simulation allows very cost-efficient exploration and ex-

SIMUTOOLS 2008, March 03-07, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.2961

perimentation with real-life network topologies and architectures. In many areas involving network analysis, *ns2* is a powerful tool; however, it also poses certain limitations to the user. Concerning the aspect of simulation efficiency for large networks, *ns2* does not out-of-the-box support a parallel execution model. It does not support generic sockets, but protocols have to be rewritten for the usage in *ns2*. Moreover, the script language interface is not intuitive to use for novice users. Alternatives to *ns2* are for example the *QualNET* simulator [13] or the Georgia Tech Network Simulator *GTNetS* [12], which, in contrast to ns2, also offers parallel execution support for large-scale simulation.

However, all these simulators have limitations in standard support of real-world network security evaluation. A lot of work has been done in the limited area of worm simulation, as described extensively by Wei et al. [16]. In this work, the authors note that existing simulators are mostly single-machine tools and hence do not scale to model realistic attack mechanisms in real-world large-scale networks. They propose 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 [17], although it allows model-driven attack tree-based simulation in a reusable object-oriented software architecture.

With *NeSSi*, we aim to provide a network simulation tool 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* follows a strict object-oriented design pattern and fosters the integration of third-party applications via a standardized socket interface. Furthermore, it provides a comprehensive Detection API for the integration and evaluation of predefined as well as external detection units. In particular, special common attack scenarios are supported and can be simulated, for example worm-spread scenarios, botnet-based DDoS attacks (see Figure 4) and many more. 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 complexity. 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 central database module. A graphical user front-end allows for real-time inspection and configuration of scenarios.

3. 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 ones. 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. As a foundation for the simulation of security scenarios, we first describe in this section the key features of *NeSSi* related to general network simulation tasks.

3.1 JIAC agent framework

NeSSi is built upon the JIAC framework [5]. JIAC is a service-centric 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 combining 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 4.

Although the software agents provide powerful application-level capabilities, their complexity unfortunately also affects the scalability of the simulation. Notwithstanding, we mitigate this problem by building upon a parallel-execution model (cf. Section 3.4). Additionally, in the distributed simulation context the agents are employed as managing entities of different parts of the simulated network domain.

3.2 Network Simulation Capabilities

At the foundation of *NeSSi* are standard network simulation features such as network and traffic generation. We will introduce the network model features in the next section, followed by a description of the traffic generation capabilities in *NeSSi*.

3.2.1 Network Model Features

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 to the network editor tab via drag-and-drop (cf. Figure 1). In the simulation, agents realize the behavior of the nodes. 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. In the central network editor window of Figure 1, such a network with a hierarchical structure is displayed. 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.

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[1]) 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

[1] http://www.eclipse.org/emf

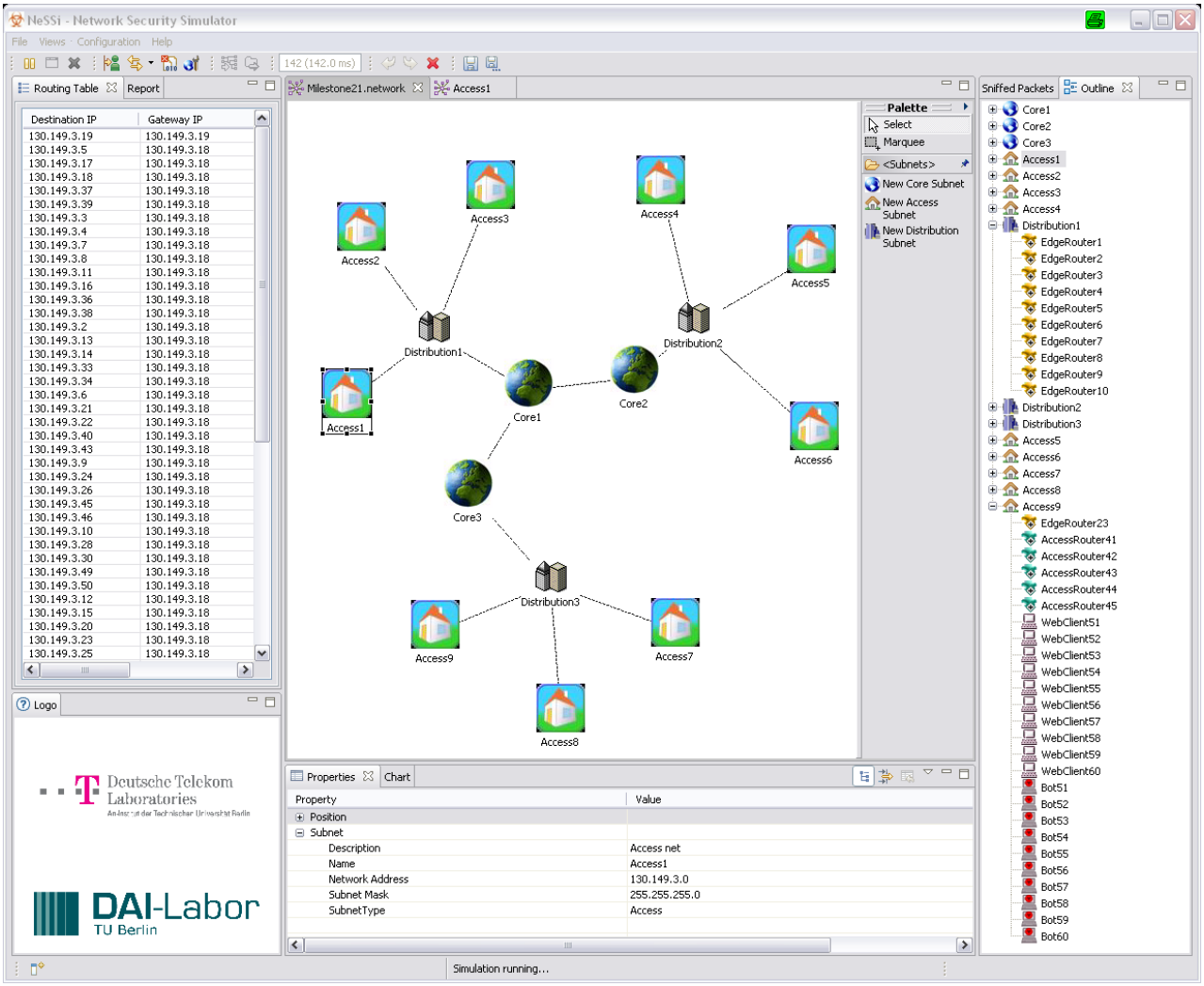

Figure 1: Graphical user interface of NeSSi

and supported routing protocols; link properties include delay and bandwidth.

NeSSi 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 based on the Dijkstra algorithm 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 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.

3.2.2 Traffic Generation

Network traffic in form of IP packets, complete with header and body, can be generated by different means. A straightforward way is to simply select a client and request a web page to be transferred (web client), or an e-mail to be sent (mail client) from a specified server. Implementing the TCP/IP protocol stack, *NeSSi* features an application layer module which is based on standard Java socket implementations. In this fashion, *NeSSi* can easily be extended to support a variety of other applications (see Section 3.3). In addition, the concept of node profiles is incorporated in *NeSSi*. Node profiles allow the customization of network node behavior in order to automatically generate traffic adhering to well-defined characteristics and using various distribution models (standard, binomial etc). According to a virtual system-wide timer, so-called profile elements can be scheduled for one-time or repeated execution. Profile elements are elementary actions, like sending single UDP packets or issuing HTTP requests. Each action contains a time stamp as well as specific context information depending on the used protocol. For example, for a HTTP request the name of the web server and the requested URL are stored in the profile element. Figure 2 shows the creation of a profile containing a HTTP request profile element.

After defining a profile, it can easily be attached to several clients by enabling it in the respective clients' context menu. This concept allows representing inherent system behavior or explicit user actions and results in the automatic generation of traffic. This also 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 mod-

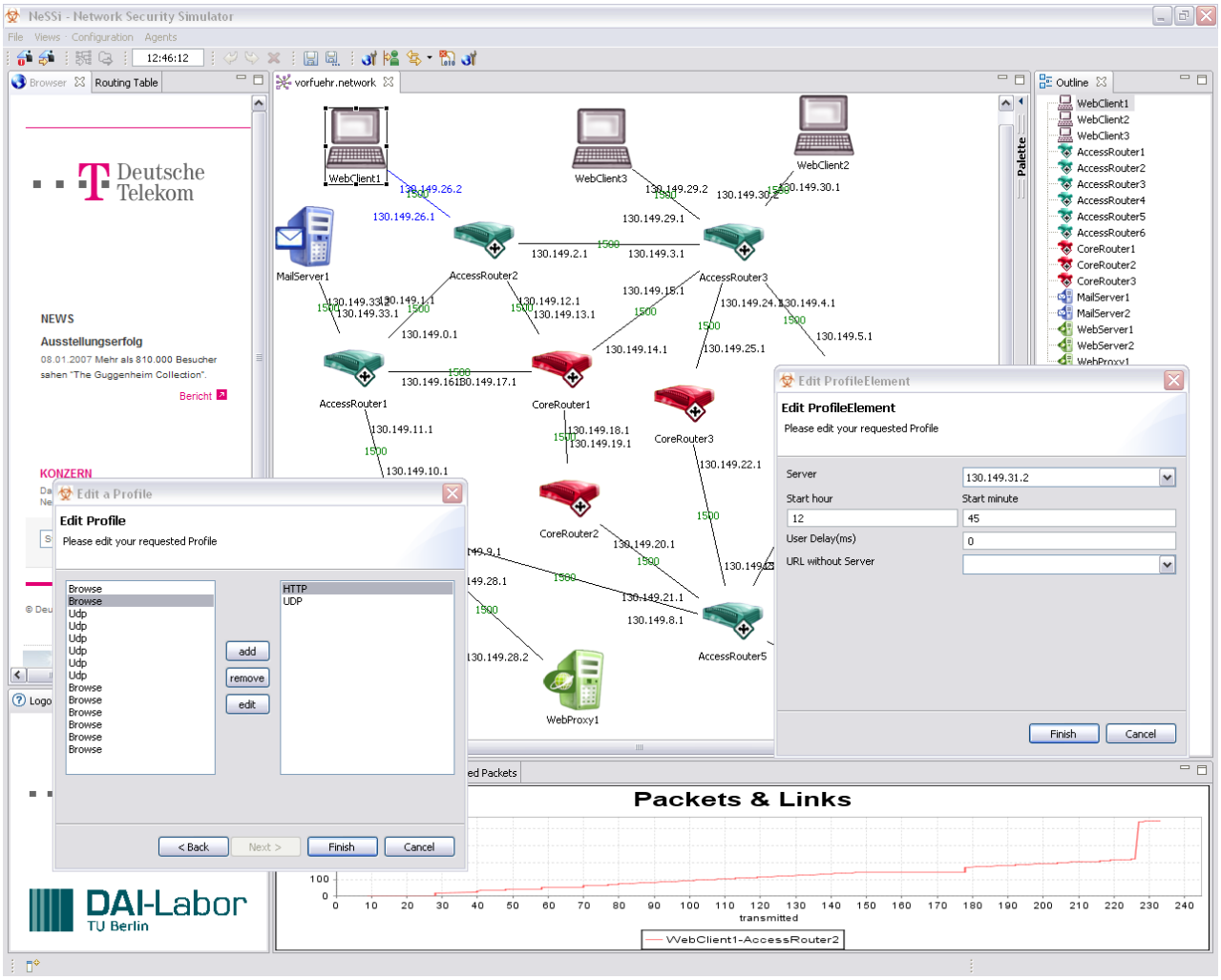

Figure 2: Detailed behavior of attached clients for custom traffic generation can be configured and scheduled

els. Automated attack generation has been successfully employed for the generating simulation data in the research of the cooperative detection approach (cf. Section 4). In the resulting paper [9], traffic statistics of captured real traffic data were mapped to node profiles.

3.3 Protocol Stack and Socket-based API

The TCP/IP reference model is the de-facto standard for Internet communication. Due 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 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 UDP and 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 in the simulation by substituting Java sockets with *NeSSi* sockets. As a proof-of-concept, the JavaMail API [2] has been successfully adapted in *NeSSi*.

All applications that are run in *NeSSi* follow a common 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*. Furthermore, the Application Layer offers an environment allowing the simultaneous execution of several application instances.

Generated traffic in *NeSSi* can also be exported to files in the *pcap*[3] format in order to inspect the data with standard traffic inspection tools such as wireshark[4]. 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

[2] http://java.sun.com/products/javamail
[3] http://www.tcpdump.org
[4] http://www.wireshark.org

summary (cf. Figure 1).

3.4 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 1, which in turn contain the end devices or routers (for example seen in Figure 2). In *NeSSi*, the handling of subnets is distributed to several processes on different individual machines, allowing the simulation to scale better with increasing network size. Figure 3 shows the architecture of the distributed simulation. On the agent level, the simulation workload distribution is as follows:

Subnet Agent (SA) A subnet agent 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 platform coordination agent (PCA). A platform is meant as a software environment for agents running on one computer. The number of subnet agents handled by the PCA is dependent on the available computing resources. The PCA receives events from the NCA and relays them to all 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 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 3.5).

GUI Agent The GUI agent acts as a mediator between the software agent environment and the graphical user interface. It transmits control information from the GUI to the agents and in turn receives data from the agents which can be displayed in the GUI, for example statistical data on traffic and encountered malware.

Network Coordination Agent (NCA) The entire network simulation is controlled by the network coordination agent (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 graphical user interface can be disconnected from the simulation environment and later reconnect to it even from another computer. Figure 3 is a graphical representation of the distributed agent architecture.

3.5 Persistent Session Management

In *NeSSi*, we refer to a scenario where we generate traffic via pre-defined profiles (cf. Section 3.2.1) 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. The database design is based on the session concept; each session consists of all traffic data that occurs in a simulated scenario between a start and end time using a dedicated network model. The network model is saved in a 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. Additionally, attack related events can be stored in the data base for evaluation purposes. Those events are explained more in detail in section 4.3.

3.6 Graphical User Interface

The graphical user interface is a Rich Client Platform (RCP) application based on the Eclipse framework. It comprises of 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. A selection of these views is:

Console Here, network status information and logged events are displayed. The desired level of verbosity can be configured.

Browser The browser view serves for displaying HTML resources which were manually retrieved by executing the respective command in the network editor.

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 editing a number of properties 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.

Chart The *Chart View* serves as a graphical depiction of aggregated statistical data collected on the network traffic for the selected link. This data is received from the PCA's databases and updated periodically.

Sniffed Packets Supplementing the chart view, individual packets can be displayed in a separate *Sniffed Packets View*. Packets are color-coded according to their protocol type and sorted by the link they were captured on. Furthermore, a summary of important information is given, such as source and destination IP address and fragmentation offset.

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 executing a pre-defined IRC exploit scenario. When the user selects it, a wizard appears, describing the scenario and guiding the user through the process.

Simulation execution can be started, paused and stopped at any time. Once a scenario was run successfully, the user can start a new scenario, opt to replay the same traffic with different detection units or deployments. When a scenario is not running, the user can also change the network topology by adding and deleting links and nodes.

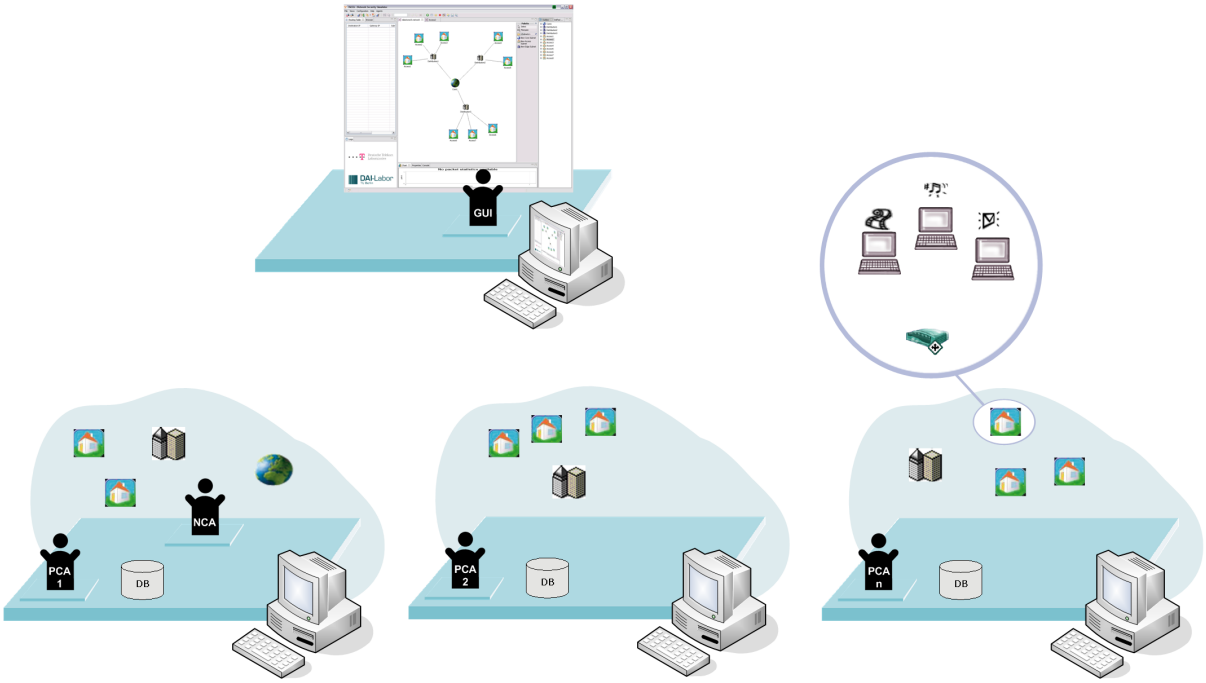

Figure 3: Architecture for distributed Simulations with NeSSi

4. SECURITY 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 5 provides a conceptual view on *NeSSi* the Network Security Simulator. 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 for the plug-in-based *NeSSi* application. On top of the bottom layer, several concepts realized in *NeSSi* can be found. These are the aforementioned general network simulation components like traffic modeling via node profiles, a parallel execution model, discrete event simulation and the visualization. Additionally, tailored to the needs of security experts, *NeSSi* incorporates the modeling of attacks and the security infrastructure (c.f. Section 4.2). Moreover, the Detection Unit API presented in section 4.1 offers the integration of detection algorithms and the applied Reporting Engine and Meta Event model enable the user to generate sophisticated statistical evaluations.

4.1 Detection Unit API

Foremost, *NeSSi* provides a *Detection Unit API* for the development of new detection algorithms as well as the integration of existing ones. The architecture for this purpose is shown in figure 6. 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 (like "every tenth packet") to narrow down the processing overhead. Accordingly, the packets are associated with time stamps.

Several detection algorithms, e.g. behavior based approaches, do not only process packets but also generate 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[5] 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 like 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 like *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.

4.2 Attack and Security-Infrastructure

The simulation setup in *NeSSi* is not only comprised of network creation (c.f. section 3.2.1) and attachment of traffic profiles (c.f. section 3.2.2), 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

[5]http://www.ietf.org/html.charters/ipfix-charter.html

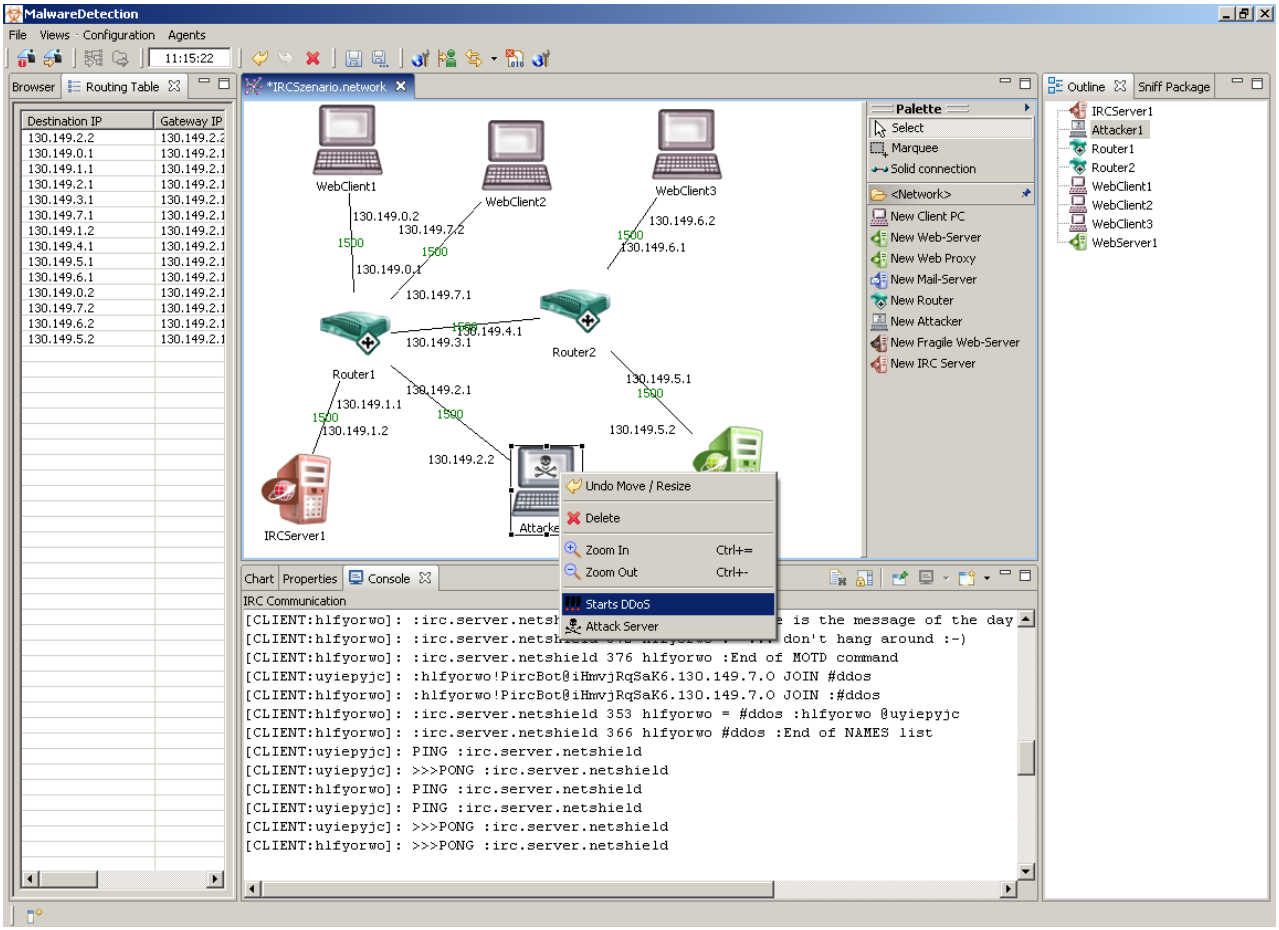

Figure 4: Simulating a DDoS attack in an IRC scenario

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 like 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 all the bots are listening to. As a result, the bots execute the desired attack.

Besides, worm propagation schemes are supported. Here, the behavior of the SQL Slammer worm and the Blaster worm has been realized exemplarily in *NeSSi*. In general, a worm propagation scheme as well as the SIR model (Susceptible Infected Recovered) 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. Additionally, 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 one can seamlessly be integrated via the application interface provided in *NeSSi* (cf. 3.3). 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 5.2. The tools used for these evaluation tasks are described in the next section.

4.3 Reporting and Evaluation

NeSSi allows the simulation of various security scenarios. Additionally, 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 has 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.

Furthermore, we apply BIRT[6] (Business Intelligence and Reporting Tools) for visualization and analysis of results. BIRT is comprised on the one hand of a graphical report designer capable of creating report templates. In those templates, different input

[6]http://www.eclipse.org/birt/phoenix

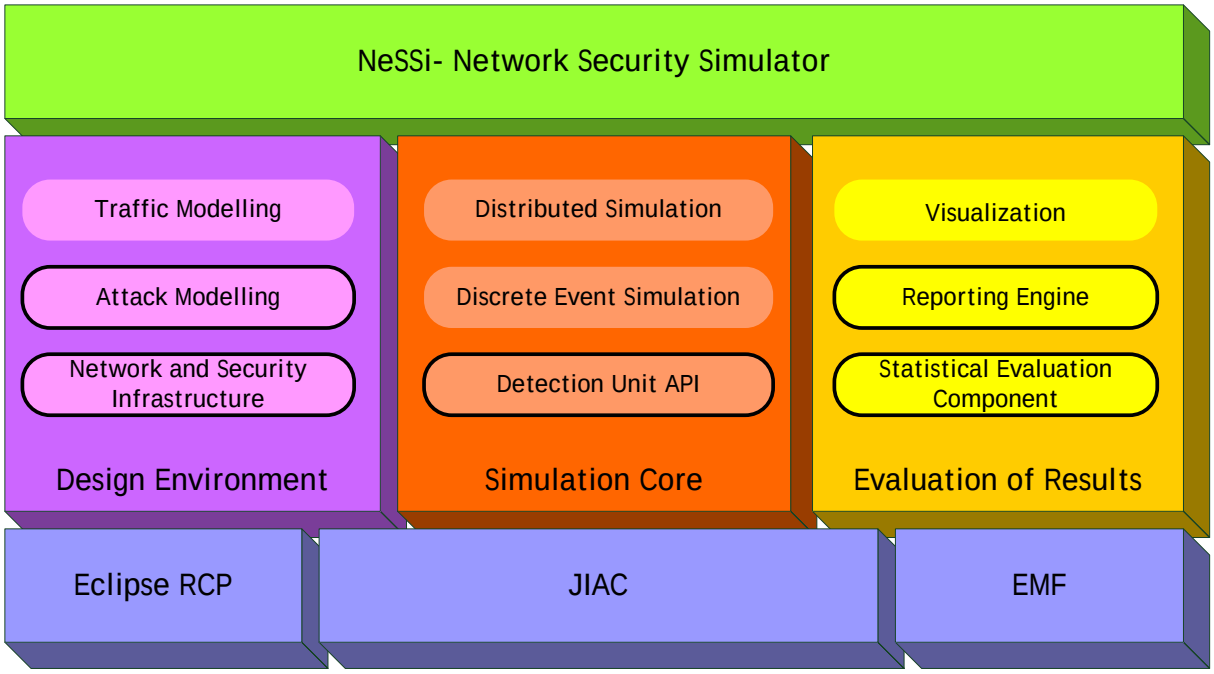

Figure 5: NeSSi, the Network Security Simulator is comprised of several components presented in the middle layer. The concepts characterizing NeSSi as a security simulation environment are accented. The bottom layer depicts some of the main used technologies.

sources like databases, simple Java objects or XML files can be declared. Additionally, 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 7 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). In the following section, practical relevance of the presented concepts is shown in some examples.

5. CASE STUDIES

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

5.1 Artificial Immune System

For example, a distributed and collaborative *Artificial Immune System* (AIS) was implemented in *NeSSi* for testing an anomaly-based detection algorithm [9]. 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 [4]. 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 to the normal data; if a detector is similar to a normal feature vector, it is deleted and a new one is created, this is done 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 [9] 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 AI Systems into detection groups.

As an extension, a scheme for the decentralized detector set generation was presented by Bye *et al.* [3]. Here, the overall feature space is partitioned in several sub spaces 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 like Symmetric Balanced Incomplete Block Design or Generalized Quadrangles a controlled level of overlap between the detector sets is realized and sets are exchanged deterministically via a peer-to-peer system. 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.

5.2 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 de-

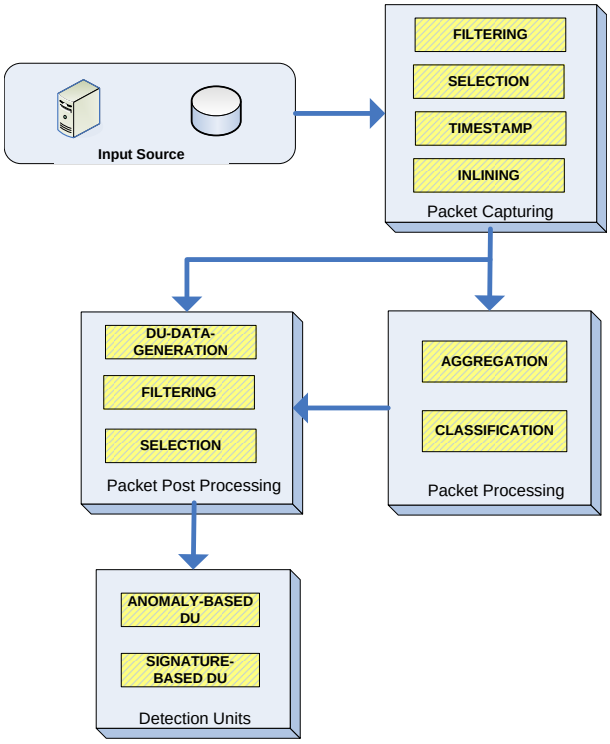

Figure 6: The components of the Detection API and the packet/data flow between them. The Packet Processing component is not necessary for all detection algorithms.

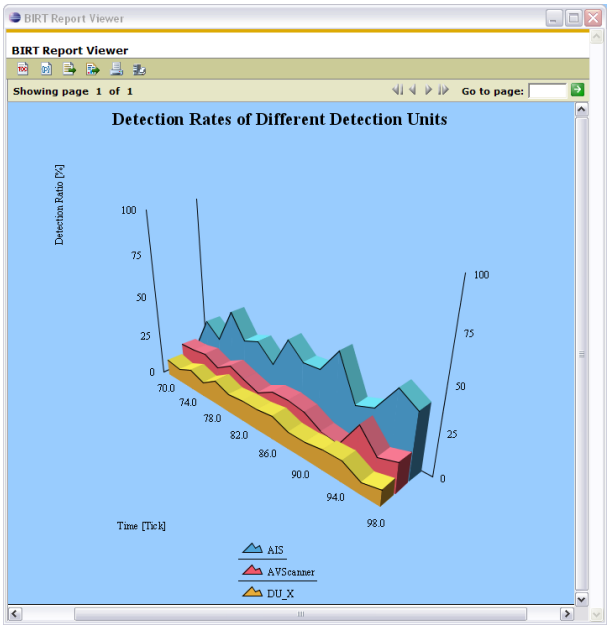

Figure 7: Reporting in NeSSi: This report shows an evaluation of the detection ratio (sensitivity) of different detection algorithms

vice placement scenario in *NeSSi*. Furthermore, the configuration dialog allows selecting 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 [2], Traffic Betweenness Centrality [1] or a selection of game-theoretic approaches [6, 14].

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 [14]. The Betweenness Centrality algorithm originates from social network theory and indicates the importance of a node in the overall network communication. It is run in *NeSSi* in an *ExecutorService* 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 [1] approach would be even more suitable.

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 simultaneously, 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.

5.3 Detection Algorithm Evaluation

In a university graduate class, *NeSSi* was used to produce data for evaluating detection algorithms implemented by students without any knowledge of the simulator. When it is not required to analyze simulation data at runtime, *NeSSi* allows to store simulation data, e.g. the IP packets, in the database. In the anomaly detection class, *NeSSi* was used to simulate "normal" 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 these different experiments were based on the simulation data stored in the *NeSSi* database.

6. 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 [11].

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 4) by peer-to-peer principles. Peer-to-peer bot nets are regarded as an emerging important threat in Network Security.

Open-source We plan to publish *NeSSi* under an open-source license.

Acknowledgement: The authors would especially like to thank their scientific advisors Ahmet Camtepe and Tansu Alpcan for their invaluable support and their fellow colleagues Thorsten Rimkus, Sebastian Linkiewicz, Marcus Lagemann and Joël Chinnow for their dedicated work on this project.

7. REFERENCES

[1] M. Bloem, T. Alpcan, S. Schmidt, and T. Başar. Malware filtering for network security using weighted optimality measures. In *Proc. of 2007 IEEE Multi-conference on Systems and Control*. IEEE, 2007. to appear.

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

[3] R. Bye, K. Luther, S. A. Çamtepe, T. Alpcan, Şahin Albayrak, and B. Yener. Decentralized Detector Generation in Cooperative Intrusion Detection Systems. In S. Masuzawa, Toshimitsu; Tixeuil, editor, *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, 2008.

[4] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself Discrimination in a Computer. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Computer Society Press, 1994.

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

[6] M. Kodialam and T. Lakshman. 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*, volume 3, pages 1880–1889, Apr. 2003.

[7] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier. Rinse: The real-time immersive network simulation environment for network security exercises. *PADS*, 00:119–128, 2005.

[8] B. Liu, D. Figueiredo, Y. Guo, J. Kurose, and D. Towsley. 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*, volume 3, pages 1244–1253, 2001.

[9] K. Luther, R. Bye, T. Alpcan, S. Albayrak, and A. Müller. A Cooperative AIS Framework for Intrusion Detection. In *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, 2007.

[10] D. M. Nicol, M. Liljenstam, and J. Liu. Advanced concepts in large-scale network simulation. In M. E. Kuhl, N. M. Steiger, F. Armstrong, and J. A. Joines, editors, *Winter Simulation Conference*, pages 153–166. ACM, 2005.

[11] ns-3 project. NS-3 network simulator. http://www.nsnam.org/docs/architecture.pdf.

[12] G. F. Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12. ACM Press, 2003.

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

[14] S. Schmidt, T. Alpcan, S. Albayrak, and A. Müller. A Monitor Placement Game for Intrusion Detection. In *Proc. of CRITIS, 2nd International Workshop on Critical Information Infrastructures Security*, Lecture Notes in Computer Science. Springer, 2007. to appear.

[15] USC Information Sciences Institute. NS-2 network simulator 2.31. HTTP://WWW.ISI.EDU/NSNAM/NS/DOC/NS_DOC.PDF.

[16] S. Wei, J. Mirkovic, and M. Swany. Distributed worm simulation with a realistic internet model. *PADS*, 00:71–79, 2005.

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