Large-scale Evaluation of Distributed Attack Detection

Thomas Gamer
Institute of Telematics
Universität Karlsruhe (TH)
Germany
gamer@tm.uka.de

Christoph P. Mayer
Institute of Telematics
Universität Karlsruhe (TH)
Germany
mayer@tm.uka.de

ABSTRACT

Evaluation of mechanisms for anomaly and attack detection is still a challenging task and hard to achieve. This especially holds for the evaluation of the large-scale behavior and efficiency of distributed detection mechanisms. Since testbeds and real networks are no feasible means for large-scale evaluation, we present in this paper a toolchain for the large-scale evaluation of distributed attack detection based on the simulator OMNeT++. Particular focus is placed on simplicity and usability of the toolchain. The interplay of the individual tools is shown by means of an exemplary attack detection. Furthermore, a performance evaluation of the individual tools is presented that shows their limitations in terms of hardware and time constraints.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*

General Terms

Simulation Environment

Keywords

Distributed Attack Detection, Anomaly Detection, Large-scale Evaluation, OMNeT++

1. INTRODUCTION

Large-scale attacks like distributed denial-of-service attacks (DDoS) or worm propagations are still part of the daily routine of the Internet. According to the Worldwide Infrastructure Security Report 2008 [1], DDoS flooding attacks with up to 40 Gbit/s of attack traffic were observed within a 12-month period starting mid 2007. During the last years, the increase of attack bandwidth has steadily outsped the increase of backbone bandwidth capacity. Thus, large-scale attacks like DDoS not only threaten victim host systems but also the networks of Internet service providers.

Promising approaches to attack and anomaly detection, respectively, are based on distributed detection or collaboration of independent detection instances [2]. Development of such distributed mechanisms and evaluation of their effectiveness and efficiency, however, requires realistic and large networks in order to actually observe the global behavior. Prototypic deployment in large testbeds is expensive and maintenance is time-consuming and complex. Deployment in real networks is difficult to achieve since isolation is necessary to a certain degree in order not to affect normal operation. Thus, simulations are a suitable and—as Ringberg et al. [14] recently pointed out—necessary alternative for large-scale evaluations.

Several simulators, e. g. OMNeT++ [17], are potentially suitable for large-scale network simulations. The results of such simulations and thus, the quality of the according evaluations heavily depend on the applied simulation environment. If, for example, the topology used is too small or does not reproduce the characteristics of real-world topologies, tested detection mechanisms may fail when deployed in real networks in spite of promising results achieved in preceding simulations. Especially in the case of attack detection, reliable results heavily depend on realistic background traffic as well as attack traffic provided within the simulations.

It is desirable that the integration of real-world attack detection systems and anomaly detection methods into a simulator be simple. This means that a real detection system at best should be transparently applicable within simulations as well as in real environments. Overall, the most important requirements for the large-scale evaluation of distributed attack detection by means of simulation are *usability*, *simplicity*, and *scalability*. It should be easily possible even for unskilled users to establish reasonable simulation environments for the purpose of evaluation. Therefore, the actual implementations should be hidden by simple graphical interfaces. Configuration of detection instances—especially in case of heterogenous networks where available instances may have different configurations—has to be scalable.

In this paper, we contribute a toolchain that facilitates large-scale evaluation of distributed attack detection by means of simulations. The advantages of our solution are:

- simulation environments as *realistic* as possible
- *transparent* integration of attack detection mechanisms
- *comparability* of implemented detection mechanisms
- *simplicity* and easy *usability*
- the tools provided are *well concerted*

OMNeT++ 2009, Rome, Italy.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552

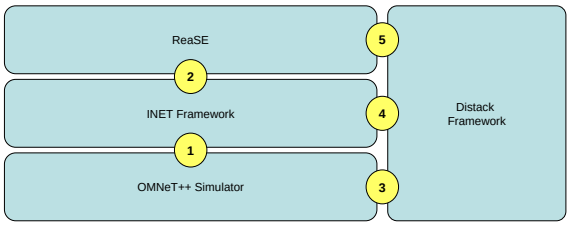


Figure 1: Outline of our simulation toolchain

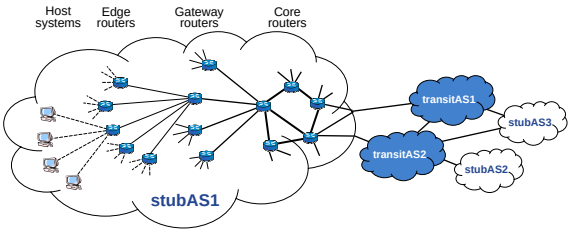


Figure 2: Hierarchical topologies created by ReaSE

The rest of the paper is structured as follows: Section 2 shortly introduces the components of the toolchain and their interrelations. Section 3 describes the graphical user interfaces that facilitate configuring the tools. Section 4 then details the interplay of the tools and presents an exemplary large-scale DDoS simulation. The performance evaluation in Section 5 shows the limitations of the simulated networks concerning hardware constraints. Finally, related work is considered in Section 6 before Section 7 gives concluding remarks and an outlook on future work.

2. SIMULATION TOOLCHAIN

The single components of the simulation toolchain we developed in order to enable large-scale simulation and evaluation of distributed attack detection are introduced in the following subsections. Figure 1 outlines the big picture of the toolchain, i. e., the components and their contact points.

The toolchain is based on one of the most popular simulators in the research area of communication networks—the discrete event simulator *OMNeT++* [17]. OMNeT++, additionally, offers many simulation models for specialized areas, e. g. the INET framework [8] that extends the simulator by a large number of Internet-specific protocols like IP or TCP as indicated by ①. *ReaSE* [7] enables generation of realistic Internet simulation environments. Contact point ② depicts that INET is extended by ReaSE through special client and server entities that allow for generation of self-similar network traffic and of attack traffic within a simulation. In addition, NED files containing realistic simulation topologies can be created. The *Distack* framework [6] provides easy integration of attack detection mechanisms with transparent support for simulators. Therefore, Distack is loaded by OMNeT++ ③. The Distack instances use protocols provided by INET, e. g. TCP sockets, to achieve a distributed attack detection ④. Furthermore, routers of the created topologies can be configured as Distack instances by ReaSE ⑤.

2.1 OMNeT++/INET

The discrete event simulator OMNeT++ [17] operates in an event-driven manner within a discrete time domain. Every action that may cause a state transition, e. g. timers or received packets, is scheduled at a particular point in time. Events then are processed consecutively in timely order. The state of the simulation environment, therefore, is assumed to be unchangeable between two consecutive events.

To simulate Internet-specific networks, OMNeT++ is used in combination with the INET framework. INET is a very popular simulation model for OMNeT++ and extends the simulator by different layers of the TCP/IP stack, e. g. network and transport layer. Furthermore, it provides Internet-specific entities like `StandardHost` and `Router`, which combine multiple layers to achieve the functionality of end and intermediate systems respectively.

A simulation in OMNeT++ is realized based on hierarchically structured *modules* that contain the simulation's functionality. *Simple modules* implement the actual functionality, e. g. the TCP or IP protocol, within one or more C++ classes. Multiple simple modules then can be connected to each other and thus, form a *compound module*. This facilitates the definition of the complete functionality of a standard host system or a router within a single compound module. The connection of modules in general is achieved by incoming and outgoing *gates*. Each module defines its gates, parameters, and submodules in a separate *NED* file. A global *NED* file finally specifies the modules and their interrelations that form the simulated network.

2.2 ReaSE

ReaSE [7] facilitates easy and repeatable creation of realistic simulation environments for a meaningful evaluation of Internet-specific systems and protocols. This also ensures that the results of different research activities can be compared with each other due to the same simulation premises. In the following, the basic characteristics and design decisions are shortly summarized—details can be found in [7].

In order to facilitate the evaluation of Internet-specific systems and protocols—with special focus on security-related topics—we identified three important aspects that must be modeled as realistically as possible:

- Internet-specific topologies,
- Self-similar background traffic, and
- Large-scale attacks.

Topology creation is divided into two hierarchical levels. First, a topology of Autonomous Systems (AS) is generated. In a second step, each AS gets a separate router-level topology (see Figure 2). The router-level topology, in turn, is structured hierarchically: it consists of core, gateway, and edge routers as well as actual host systems. Communication between different AS actually takes place between core routers only. Both AS-level and router-level topology have to show a power-law distribution in node degree according to [20]. In case of router-level, however, additional aspects like market demands, link costs, and hardware constraints [11] are considered.

If aiming for realistic simulations, not only traffic of the evaluated protocol or system has to be generated and examined. Additionally, it is necessary to create *background traffic* showing the same characteristics as normal traffic in real

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552

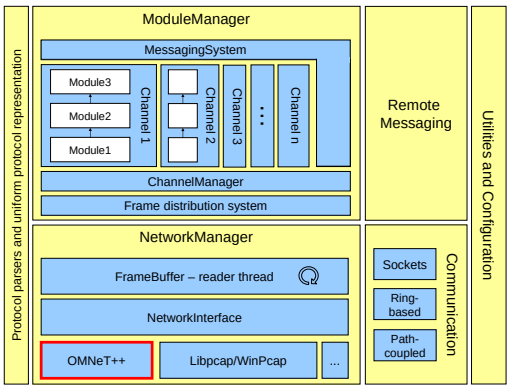


Figure 3: Simplified Architecture of Distack

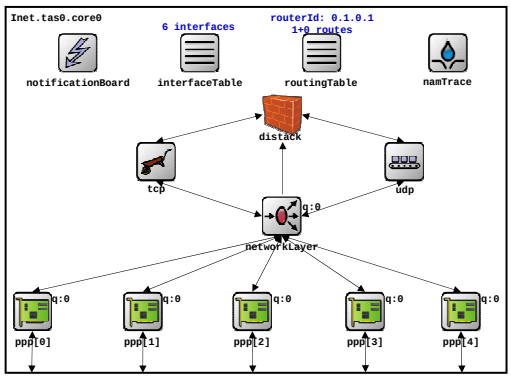


Figure 4: A Distack-enabled Router in OMNeT++

networks does. Such traffic in most cases heavily influences behavior of the observed protocol or system. Thus, ReaSE extends INET by special client and server entities. These entities generate a reasonable mix of different traffic types—e. g. web, streaming, mail, and ping traffic. The aggregated traffic then shows self-similar behavior as traffic in real networks does [3]. Bearing in mind security-related topics, a further aspect has to be considered in order to achieve realistic simulation environments: generation of attack traffic. Using ReaSE, a researcher can generate distributed denial-of-service (DDoS) attacks based on the mechanisms of Tribe Flood Network [5]—a notorious DDoS tool used in the wild. Furthermore, UDP or TCP based worm propagations, which rely on a simple probing mechanism as used with Code Red I [21], can be easily used in simulations.

Usage of traffic trace files is not considered in this work due to the large number of nodes that should be simulated. This results in the necessity to record traces at multiple different spots in the Internet, which is difficult to achieve.

2.3 Distack

The *Distack Framework* [6] has been designed to allow for easy development and evaluation of attack detection and traffic analysis mechanisms. It provides an environment that enables implementation and combination of various mechanisms. This ensures easy comparability of different mechanisms. In addition, transparent deployment of implemented mechanisms in real as well as simulation environments is provided. Section 2.3.1 summarizes the architecture of the Distack framework. Section 2.3.2 describes how Distack is actually integrated into OMNeT++.

2.3.1 Architecture

Figure 3 shows a simplified version of the Distack architecture. The *NetworkManager* provides a unified abstraction from the underlying network and runtime environment—e. g. real-world systems with a packet capture interface like Pcap or simulated systems using OMNeT++. This ensures transparent deployment in various runtime environments with only little changes.

On top of the NetworkManager the *ModuleManager* enables to dynamically load user-written lightweight *building blocks* at runtime. These building blocks—in the following called *modules*—are shared libraries that perform e. g. packet inspection, filtering and sampling [22], or various anomaly detection methods. They are configured and virtually instantiated with a unique name by an external XML-based *configuration file*. Hereby, a module can be used multiple times simultaneously with different configurations. In addition, modules can be flexibly combined to a so called *channel* to achieve more complex functionality. Channels are also defined in the configuration file and contain a sequentially ordered set of modules. Incoming network packets are consecutively given to each loaded channel, traversing all modules in the channel. Each module, therefore, is able to process the packet and perform its operations.

As modules are lightweight and self-contained building blocks, the need for information exchange between modules may arise. This is provided by the data-centric *MessagingSystem*. Modules register for a particular type of information that is delivered to them once another module sends this type of information into the MessagingSystem. To allow for remote communication with other Distack instances, the MessagingSystem is extended by the *RemoteMessaging* component. Different communication mechanisms—e.g. sockets or ring-based signaling—are offered by the *Communication* component. The selected mechanism then performs the actual data transmission.

2.3.2 Integration into OMNeT++

In order to integrate Distack into OMNeT++ the complete framework is encapsulated into a `cSimpleModule`. The functionality of the Distack `main()` method is moved into the method `cSimpleModule::initialize()`. Furthermore, Distack is compiled into the shared library `libdistack.so`. This library can be dynamically loaded by OMNeT++ at runtime using the `load-libs` command in *omnetpp.ini*.

Distack builds on the compound modules `Router` and `NetworkLayer`, which are provided by ReaSE in a slightly modified version, to host the Distack `cSimpleModule`. Such a modified `Router` creates copies of all received packets and passes them up to Distack for processing. Figure 4 shows the composition of such a Distack-enabled router that also provides communication methods based on UDP and TCP.

Further abstractions are provided that help researchers to easily use Distack in different runtime environments like real-world systems and simulators. Examples for such abstractions include access to routing tables, or transparent

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552

use of timer functionality in real-world as well as time discrete simulation environments. Furthermore, different representations of network packets are handled by Distack in a unified way and are transparent for modules. For detailed description on how we integrated a real-world application like Distack into OMNeT++, see [12].

2.4 Distributed Attack Detection

The Distack components *RemoteMessaging* and *Communication* allow for collaboration of various detection instances. Distributed attack detection often is used to improve the detection quality and efficiency as opposed to local observation only. The RemoteMessaging performs message serialization and specifies a PDU format for the transport of the serialized messages. The Communication component provides a generic interface for actually sending a PDU. At the time Distack was published [6], it provided only one communication method for real networks: TCP sockets. The implementation of a path-coupled signaling method was intended in the future. These methods, however, cannot be applied within a simulator like OMNeT++ since all Distack instances are simulated on a single system and thus, no real communication is applicable. Therefore, we extended Distack by further communication methods that are usable within a simulator: an equivalent method to real TCP sockets as well as path-coupled and ring-based signaling.

In order to model real TCP sockets in OMNeT++ we implemented a communication method that uses the TCP functionality of the INET framework in order to open a TCP connection to a remote Distack instance. As with real sockets, the destination addresses of remote instances must be specified in the Distack configuration. This allows for distributed mechanisms between a fixed number of preconfigured instances, e. g. master/slave approaches like [18]

The *ring-based* communication method we implemented offers an inbuilt mechanism for the dynamic discovery of neighbor instances. With that method, detection instances collaborate per unicast with each neighbor instance that is a maximum of n IP hops away. The neighbors are discovered by sending a broadcast request message with a time-to-live value set to n. For the simulator we simplified the original procedure by introducing a *RingManager*, at which every Distack instance registers during simulation startup. The RingManager calculates the neighbors of each instance that then establishes a TCP connection to all its neighbors. Thus, we abstract from the actual discovery method and do not simulate its message exchange.

Using path-coupled collaboration, in-network instances can propagate information on detected DDoS attacks to upstream instances. These instances then are able to detect further flows of this specific attack easily. In case of undirected attacks, path-coupled downstream signalling may facilitate tracing the attackers. The *path-coupled discovery* is the most complex method we implemented for OMNeT++. It is based on a prior version of the GIST protocol [16], which is an up-to-date signalling transport protocol suggested by the IETF NSIS working group. The actual implementation assumes less simplifications than ring-based discovery. In order to find a path-coupled neighbor, a UDP query message is sent in direction of a particular destination, e. g. the victim of a DDoS attack. In addition, the Router Alert Option (RAO) [9] of this message is set. This ensures that other Distack instances on the path to the destination process the query message. The neighbor instance then establishes a TCP connection to the requesting instance after processing of the request message. The exchange of signaling information finally is performed using the TCP connection.

The communication method that actually should be used in a particular simulation has to be specified by the user in the Distack configuration.

3. FULFILLING THE REQUIREMENTS

As we already mentioned in Section 1, the main requirements for a toolchain that provides large-scale evaluation of distributed attack detection are simplicity, usability, and scalability of configuration. These are necessary to enable usage of our toolchain also for unskilled users and thus, to facilitate a wide-spread usage of repeatable and comparable large-scale evaluations. In order to achieve usability and simplicity, we decided to offer graphical user interfaces (GUI) for the generation of realistic simulation environments as well as the configuration of detection instances.

Generation of realistic environments.

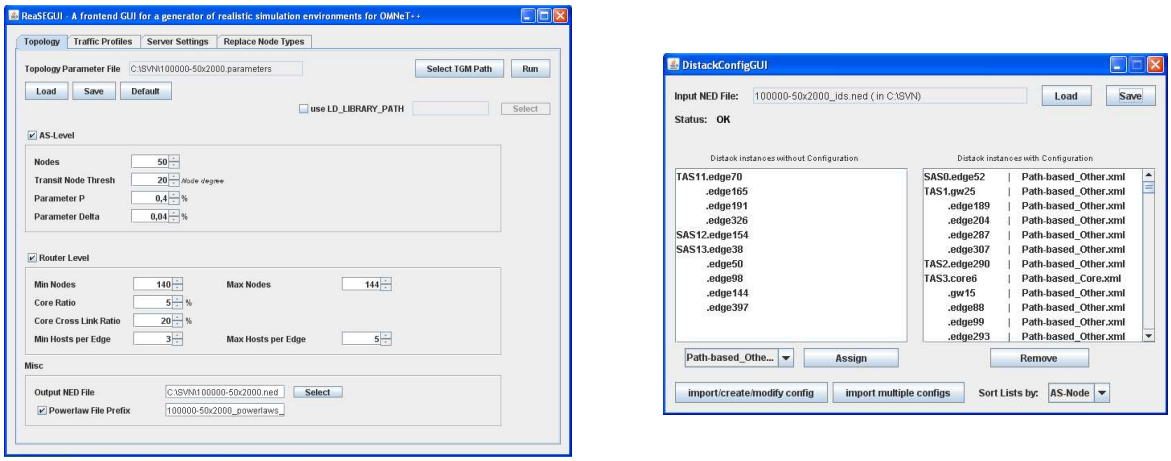
At the time ReaSE was published [7], it consisted of multiple command line tools that deal with a different task each, e. g. topology creation or integration of server entities. In addition, parameter files are necessary for some of the tools as well as for traffic generation during the simulation. For users that have no experience with ReaSE the process of generating a realistic simulation environment using the available tools could be confusing. Thus, this process may be error-prone and time-consuming. Therefore, we developed a graphical tool that unifies all necessary tasks in a single user interface. The GUI hides the different command line tools, which are actually invoked, from the user. This means that the GUI supports transparent creation of the required parameter files and execution of the various command line tools. This ensures simplicity and usability of ReaSE.

Figure 5(a) shows the main tab of the ReaSE GUI. This tab facilitates input of all parameters necessary for topology generation. In addition, it suggests default values if required and finally creates the parameter file before running the according command line tool in the background. Furthermore, traffic profiles are needed during a simulation in order to generate self-similar network traffic. Such traffic profiles can be created, deleted and modified in a further tab. Having finished the definition of the profiles, they are saved to a parameter file that can be included into the *omnetpp.ini* file for usage during a simulation. Two more tabs exist for integration of server and client entities as well as special entities like DDoS zombies or Distack instances.

Configuration of heterogeneous detection instances.

The configuration of a Distack instance is based on an XML-based configuration file, which contains information about general settings as well as module and channel definitions. In the case of distributed detection mechanisms multiple detection instances are present within the simulation environment. Especially in simulations that involve a large number of detection instances, manual assignment of configurations is time-consuming and dull work. Often, usage of heterogenous configurations is reasonable. Core routers e. g. have to analyze larger traffic volumes than edge routers. Thus, it makes sense to collectively configure all core routers

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552



(a) Internet-specific Topology

(b) Configuration Assignment for OMNeT++

Figure 5: ReaSE Configuration GUI (a) and Configuration GUI for Distack detection instances (b)

with lower sampling probabilities. In case heterogeneous configurations are needed,, manual creation of different configurations and assignment to detection instances becomes complex and error-prone. Therefore, we developed a GUI (see Figure 5(b)) that provides a quick and clear overview of present detection instances and enables an easy assignment of available configuration files to the present instances. Scalability of configuration is ensured since assignment of a single configuration to multiple instances is easily possible. In addition, the GUI provides different alternatives of displaying the present instances, e. g. aggregated by the Autonomous System they belong to or by their specific role. This further simplifies the task of configuration assignment.

Another window of the GUI that can directly be accessed from the main window manages creation and modification of configuration files. This ensures that even unskilled users know all possible configuration parameters, their current values, and each possible alternative at a glance and thus, are able to generate their intended configuration file easily.

4. SIMULATION ENVIRONMENT

In Section 4.1 we detail on the interplay of the toolchain introduced in Section 2. Section 4.2 then describes a DDoS attack simulation based on a simple exemplary scenario.

4.1 Toolchain Interplay

Attack detection simulation using our toolchain is a well-defined process that we will shortly describe in this Section.

1. An initial setup involves installation of OMNeT++ and INET. Afterwards, ReaSE can be set up.
2. Having installed these simulation tools, the Distack Framework can be compiled with simulation support.
3. The first two steps represent one-time setup. Now, a network topology and traffic profiles can be created by using the GUI of ReaSE (see Section 3). This step results in a *NED* file describing the simulated network and in a traffic profile parameter file. The resulting NED file contains not only routers, servers and clients but also includes a configurable percentage of special entities like Distack instances or DDoS zombies. This way, easy integration of Distack instances is achieved.
4. In a next step the attack detection mechanisms have to be implemented as Distack modules. They can later be applied in real-world environments, too.
5. Configuration files for Distack instances have to be created or modified, respectively. Each Distack instance then either can be assigned a unique configuration, or groups of Distack instances can be assigned the same configuration. Using the provided GUI (see Section 3) for configuration assignment makes large-scale configuration easier and more scalable.

4.2 Exemplary DDoS Attack Detection

We now detail on an exemplary DDoS simulation we performed using the toolchain introduced in this paper. Our simulation network consists of 20 AS. Each AS contains 4 core routers that form the interconnections between the different AS. The core routers also connect 1–4 gateway routers each, which each in turn connect 4–25 edge routers. 3–5 actual hosts are finally connected to the edge routers. This results in an average number of 522 entities per AS, making up a total of 10 440 entities in the simulation. About 76 hosts have been randomly replaced by DDoS zombies that collectively launch an attack at a web server victim. The victim resides in Transit AS 0 (`tas0`) and is attached to `tas0.edge13`, `tas0.gw4`, and finally `tas0.core0`. At simulation time 1 600 s the DDoS zombies start the DDoS flooding attack based on TCP SYN packets. Each DDoS zombie first decides if it takes part in the attack. If so, it starts sending TCP SYN packets with a fixed rate to the victim. Optionally, spoofing of source IP addresses can be used. There are, however, more attack types that can be used for simulation of a DDoS attack, e. g. a UDP flooding with user-specified packet sizes. To achieve a realistic ramp-up behavior each DDoS zombie delays its start for a uniformly distributed time between 0 s and 60 s.

A default traffic profile is used to generate background traffic for the simulation. Routers on the above-mentioned path towards the victim inside `tas0` are replaced by Distack instances to monitor the network traffic. Each Distack

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552

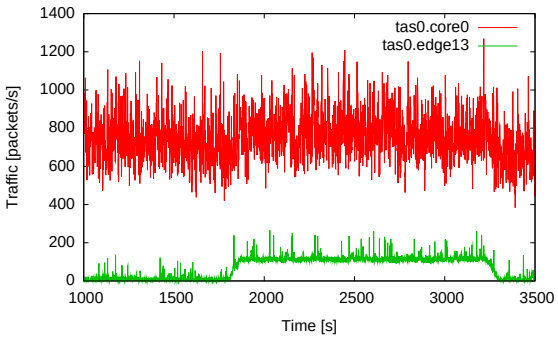


Figure 6: Monitored network traffic

instance only loads one simple module that gathers information about the number of packets observed per second[1].

Figure 6 shows the monitored traffic in packets per second at two of the Distack instances in the network. Both routers with Distack instances reside in the victim AS `tas0`. It can be seen, that the attack traffic is observed best near the victim, i. e., on `tas0.edge13`. On `tas0.core0`, the attack is not visible since too many flows aggregate here. This shows the challenge of detecting attacks by local observation only inside the network, e. g. on `tas0.core0`. Collaboration of distributed attack detection instances, therefore, promises improvement of detection.

5. PERFORMANCE EVALUATION

In this section we will give some insights into the performance evaluations of ReaSE and Distack. In order to provide users of the toolchain with a feeling about the basic behavior of the single tools, we measured performance parameters like memory usage, duration of simulations, and messages generated by OMNeT++ during a particular simulation. The simulations were carried out on a 64-bit Ubuntu Linux operating system with 2.6.24-21 kernel. The hardware consisted of an Intel Xeon 5160 dualcore CPU with 3 GHz and 4 Mb shared L2 cache as well as 32 GB of RAM. The simulation results are based on OMNeT++ 3.4 and the according INET framework.

5.1 ReaSE

For the performance evaluation of ReaSE we first created various topologies (see Table 1) that differ in size, i. e., the total number of simulated systems, and in the number of Autonomous Systems (AS). The router-level topology of each AS within a particular topology was generated with fixed parameters. Due to the random numbers used by the generation algorithm [11], this however does not mean that the router-level topologies all are equal. The first topology of Table 1, for example, has about 1 000 nodes in total and consists of 5 AS with a size of about 200 router-level nodes each. In the following, we use the term T-A$x$$R$ to indicate a topology with about T nodes in total. The topology consists of A AS, which contain about R router-level nodes each. All simulations of the topologies listed in Table 1 were run with

[1]This is a highly simplified scenario. Usually, Distack instances perform actual attack and anomaly detection that is not within the scope of this paper.

Topology size	# AS x # routers per AS			Seeds
1 000	5x200	10x100	20x50*	20
5 000	10x500	20x250	50x100	20
10 000	10x1000	20x500*	50x200	10
50 000	20x2500*	50x1000	100x500	5
100 000	20x5000	50x2000	100x1000	2

Table 1: Topologies used for evaluation

Topology	Min	Max	Average	Conf. Interval
1000-20x50	19	89	48.4	8.72
10000-20x500	406	570	499.5	18.15
50000-20x2500	2 152	2 422	2 264.3	36.65

Table 2: Numbers of router-level nodes per AS

multiple seeds. For the values marked with an asterisk additional topologies were generated based on the same parameters. During the simulations, normal background traffic was created using multiple traffic profiles. Please refer to [7] for details on the traffic profiles that are provided by ReaSE.

First, we examined the behavior of the generation algorithm for router-level nodes. We generated each topology marked with an asterisk 6 times and examined the number of nodes within each AS. Table 2 shows the statistical values we derived for one representative generation per topology: the minimal and maximal number of router-level nodes in regard to all 20 AS as well as the average number of nodes over all 20 AS. In addition, we calculated the 95 % confidence intervals of the averages. Observing the three topology sizes we noticed that the average value of the router-level sizes becomes significantly more stable with increasing router-level size, which is shown by the relatively decreasing confidence intervals. This stabilization is caused by the fact that only the router entities rely on a power-law distribution. Hosts, which form the largest part of the router-level topology, rely on a uniform distribution.

In a second step, we ran simulations based on all the topologies presented in Table 1 and examined the memory usage and duration of all simulations. Memory usage was measured based on the *proc* filesystem by observing the virtual size of the process in kB including code, data, and stack. Simulation duration resembles the cumulative CPU time of the process in seconds. Since each topology was simulated with multiple seeds, we averaged the performance values of all runs and calculated the according 95 % confidence intervals. Figure 7(a) shows the resulting averages and confidence intervals for all topologies. Columns 2 and 3 of Table 3 show the calculated averages of memory usage and duration for one exemplary topology per topology size. We observed that confidence intervals are negligibly small for the topologies up to 10 k nodes. Topologies with 50 k and 100 k nodes were simulated with less seeds. Thus, the larger confidence intervals shown in Figure 7(a) should further decrease with additional simulations.

The topologies marked with an asterisk, which were generated 6 times, were additionally simulated with multiple seeds each in order to examine if the random generation of router-level topologies influences the performance values. The results, however, showed no significant deviation from the previously presented values. In summary, all

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552

Topology	Memory	Duration	Created	Present
1k-10x100	87 813	47	11 698 k	36 k
5k-20x250	370 052	305	52 268 k	175 k
10k-20x500	738 478	660	94 483 k	262 k
50k-50x1000	3 277 228	5 074	521 281 k	1 347 k
100k-50x2000	6 934 726	10 270	995 345 k	2 516 k

Table 3: Exemplary performance values of ReaSE

# Distack	Memory	Memory per Distack
0	744 687	–
1	745 907	1 220.00
2	745 958	635.50
10	746 256	156.90
50	747 767	61.60
2 289	816 200	31.24

Table 4: Memory usage of Distack instances

the results we obtained from our simulations clearly show that both performance values memory usage and simulation duration increase almost linearly with topology size. The small discrepancy is caused by the fact that the number of events per seconds that can be processed by OMNeT++ is not totally independent of topology size but decreases from 1 314 k in case of 1000-10x100 topology to 583 k in case of 50000-50x1000 topology.

Finally, we considered the number of messages that were created in total by OMNeT++ during a simulation and the average number of present messages as performance measures. Figure 7(b) again shows the averages and confidence intervals for all topologies. In addition, columns 4 and 5 of Table 3 list the calculated averages for one exemplary topology per topology size. From these values we can see that both, created and present messages, increase linearly with topology size, e. g. about 11 million of created messages in case of 1 000 nodes and about 1 billion in case of 100 000 nodes. This means that all performance values increase linearly with topology size. Thus, in summary, the available memory is the main limitation for the simulated topologies.

A last thing we realized during performance evaluation is that in small topologies not all available traffic profiles are used. This is caused by the fact that for some profiles no corresponding server exists in the topology due to the small number of routers. If the traffic profiles of non-existent servers are not deleted from the parameter file before startup of the simulation, the simulation does sometimes not behave as expected, i. e., performance values may differ significantly from the averages previously presented.

5.2 Distack

Resource consumption of Distack instances is of special interest in large-scale simulations. As shown in Section 5.1, memory usage can rise drastically and become a critical parameter that finally determines the upper bound of topology size. Adding Distack instances to a simulation increases memory usage. This additional memory consumption should be small in order to facilitate large-scale simulations.

Table 4 shows the memory usage in kB of a 10k-20x500 topology containing a varying number of Distack instances as well as the additional memory usage in kB of each Distack instance in comparison to the reference simulation without detection instances. Each setting was simulated with 5 seeds that show negligible differences in memory consumption. The Distack instances use only a single module that performs basic statistical traffic monitoring. No collaboration exists between detection instances. We can see from the third column of Table 4 that the additional memory consumed by each Distack instance is decreasing. This is caused by the fact that Distack instances can share objects because they run in the same process space during a simulation. One example for such shared objects is a memory pool that Distack uses for fast packet parsing. Sharing such objects between instances enables efficient memory management and lowers memory usage of each Distack instance. The increased memory consumption when deploying 0 Distack instances in comparison with Table 3 results from the fact that Distack and required libraries are loaded into memory through OMNeT++, but no Distack instance is created.

6. RELATED WORK

Related work in regard to generation of realistic simulation environments and to integration of real attack detection systems into a simulator has already been described in our previous work [6, 7]. There are existing topology generators like BRITE [13] as well as multiple traffic generators like TrafGen [4] that are able to generate simulation environments. Most topology generators, however, are not applicable with OMNeT++ or do not consider recent research results. The traffic generators often are not scalable for large-scale simulations since behavior of all nodes has to be configured manually. Snort [15] or PreludeIDS [19] are just two examples for existing attack detection frameworks. For none of the frameworks known to us, however, efforts have been made to port them to a simulation environment. Thus, a huge overhead would be necessary to integrate them into a simulator.

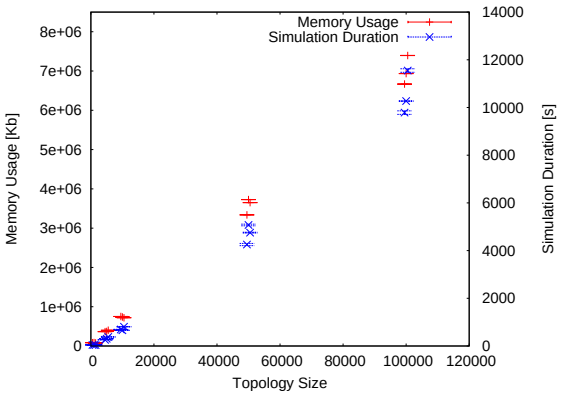
To the best of our knowledge, there is only one approach that seems to offer a possibility for large-scale evaluation of attack detection in simulation environments and that considers all necessary aspects: DDoSSim [10]. This publication, however, does not provide a detailed description of the functionality. In addition, no implementation of this tool or additional information is freely available.

7. CONCLUSION AND OUTLOOK

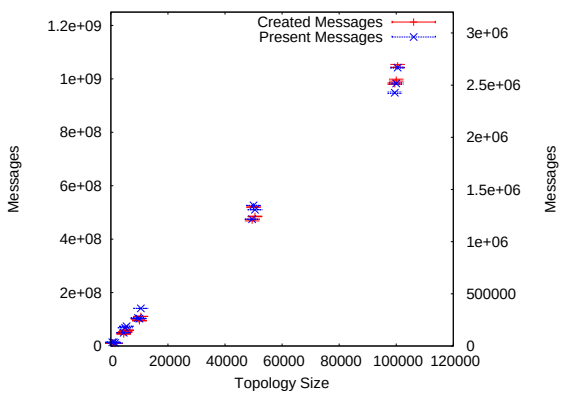
In this paper, we presented a toolchain that builds on the discrete event simulator OMNeT++ and its INET framework to ensure comparable and repeatable results without high costs and maintenance overhead. The tool *ReaSE* extends the basic simulation engine by generation of realistic simulation environments. Easy integration of real attack detection into the simulator as well as support for distributed detection mechanisms is provided by the tool *Distack*. Usability and simplicity of the proposed toolchain are ensured by providing graphical user interfaces. A performance evaluation of the tools showed that hardware requirements for the simulations and the simulation duration increase only linearly with the topology size. In summary, the toolchain establishes a basis for a large-scale evaluation of distributed attack detection systems.

In future work we will perform an evaluation of the attack identification and distributed collaboration of detection in-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552



(a) Memory usage and simulation duration



(b) Created and present messages

Figure 7: Performance evaluation of ReaSE

stances we developed. Future work, however, also includes further extension of the toolchain. Integration of additional attacks and variations of already implemented attacks like DDoS are necessary. In addition—since both tools are available as open source software—we hope that other researches contribute their detection mechanisms and the attacks they used for evaluation. This ensures even more reasonable evaluation results if attacks modeled by others are used to test the own detection mechanisms and vice versa. Finally, we are currently working on the migration to OMNeT++ 4.0[2], which should be finished in near future. Currently, we are also working on a tool for real-time visualization of a detection instance's state.

8. ACKNOWLEDGMENTS

We would like to thank Claus Faller and Melvin Williams for their valuable help in the evaluation part of this work.

9. REFERENCES

[1] Arbor Networks. Worldwide Infrastructure Security Report. http://www.arbornetworks.com/report, Oct. 2008.

[2] Y. Chen, K. Hwang, and W.-S. Ku. Collaborative Detection of DDoS Attacks over Multiple Network Domains. *IEEE Trans. Parallel Distrib. Syst.*, 18(12):1649–1662, 2007.

[3] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, Dec. 1997.

[4] I. Dietrich. OMNeT++ Traffic Generator, Sept. 2006. http://www7.informatik.uni-erlangen.de/∼isabel/omnet/modules/TrafGen/.

[5] D. Dittrich. The "Tribe Flood Network" distributed denial of service attack tool, Oct. 1999. http://staff.washington.edu/dittrich/misc/tfn.analysis.

[6] T. Gamer, C. P. Mayer, and M. Zitterbart. Distack—A Framework for Anomaly-based Large-scale Attack Detection. In *Proc. of the 2nd SECURWARE conference*, pages 34–40, Aug. 2008. Available at *http://www.tm.uka.de/distack*.

[7] T. Gamer and M. Scharf. Realistic Simulation Environments for IP-based Networks. In *Proc. of the 1st OMNeT++ Workshop*, pages 1–7, Mar. 2008. Available at *http://www.tm.uka.de/rease*.

[2]Thanks to Andras Varga for the great help in doing the basic migration work.

[8] INET Framework. http://www.omnetpp.org/pmwiki/index.php?n=Main.INETFramework, Sept. 2007.

[9] D. Katz. IP Router Alert Option. RFC 2113, IETF, Feb. 1997.

[10] I. V. Kotenko and A. Ulanov. Simulation of Internet DDoS Attacks and Defense. In *Proc. of ISC*, pages 327–342, Oct. 2006.

[11] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the internet's router-level topology. In *Proc. of ACM SIGCOMM*, pages 3–14, Sept. 2004.

[12] C. P. Mayer and T. Gamer. Integrating real world applications into OMNeT++. Telematics Technical Report TM-2008-2, Universität Karlsruhe (TH), Feb. 2008. ISSN 1613-849X.

[13] A. Medina, I. Matta, and J. Byers. BRITE: A Flexible Generator of Internet Topologies. Technical Report 2000-005, Boston University, Jan. 2000.

[14] H. Ringberg, M. Roughan, and J. Rexford. The Need for Simulation in Evaluating Anomaly Detectors. *SIGCOMM Computer Communication Review*, 38(1):55–59, Jan. 2008.

[15] M. Roesch. Snort. http://www.snort.org, 2001.

[16] H. Schulzrinne and R. Hancock. GIST: General Internet Signalling Transport. Internet draft, IETF, Oct. 2008. Work in Progress.

[17] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of the 15th ESM*, pages 319–324, June 2001.

[18] G. Vigna, F. Valeur, and R. Kemmerer. Designing and Implementing a Family of Intrusion Detection Systems. In *Proc. of 9th European Software Engineering Conference*, pages 88–97, Sept. 2003.

[19] K. Zaraska. Prelude IDS: current state and development perspectives. http://www.prelude-ids.org, 2003.

[20] S. Zhoua, G. Zhang, G. Zhang, and Z. Zhuge. Towards a Precise and Complete Internet Topology Generator. In *Proc. of ICCCAS*, volume 3, pages 1830–1834, June 2006.

[21] C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis. *Proc. of the 9th ACM conference on Computer and communications security*, pages 138–147, Nov. 2002.

[22] T. Zseby, M. Molina, F. Raspall, N. G. Duffield, and S. Niccolini. Sampling and Filtering Techniques for IP Packet Selection. Internet draft, IETF, July 2008. Work in Progress.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5552
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5552