Realistic Simulation Environments for IP-based Networks

Thomas Gamer
Institut für Telematik,
Universität Karlsruhe (TH)
Germany
gamer@tm.uka.de

Michael Scharf
Institut für Telematik,
Universität Karlsruhe (TH)
Germany
s_scharf@tm.uka.de

ABSTRACT

During development of new protocols and systems researchers in most cases use simulations for evaluation of their product, especially in the area of communication networks. The quality of the simulation environment, however, significantly influences the quality of a product's evaluation. Therefore, simulation environments as realistic as possible are necessary in order to get reliable results. In this paper we present *ReaSE*, a tool for creation of such realistic environments. It considers multiple aspects: topology generation – on AS level as well as on router level –, traffic patterns, and attack traffic. Furthermore, *ReaSE* is based on current state of the art solutions.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*

General Terms

Simulation Environment

Keywords

Power Law-based AS Topologies, HOT-based Router Topologies, Traffic Profiles, DDoS Attack Traffic, OMNeT++

1. INTRODUCTION

Most research projects in academic as well as industrial settings all over the world use simulations in order to evaluate their products before publishing or introducing them into the market. The results of such simulations and thus, the quality of the according evaluations, too, heavily depend on the applied simulation environment, e.g. which network topologies or traffic patterns are used for the simulations. If the topology used, for example, is too small or does not reproduce the characteristics of real-world topologies, the developed product may fail after deployment in real networks in spite of promising results achieved in preceding simulations. Traffic patterns are another important aspect in regard to network simulations, e.g. if an intrusion detection system has to be evaluated reliable results heavily depend on realistic background as well as attack traffic provided within the simulations.

OMNeT++ [17] is one of the most popular simulators in the research area of communication networks. It is based on the simulation of discrete events. Additionally, it offers a lot of extensions for specialized areas, e.g. the INET framework [1] that supports simulation of the Internet, i.e. MAC, IP, and transport layer, or the mobility framework [2] that enables simulation of mobile ad-hoc networks. Amongst others, OMNeT++ is used to evaluate new networking protocols, mechanisms and algorithms in large networks since a prototypic deployment of such research products in large testbeds or in real networks in most cases is too complex and too costly.

In order to perform an evaluation based on a network simulator a simulation environment is required that defines the basic conditions of the simulation. Keeping the research area of communication systems in mind, a simulation environment must define the topology of the simulated network as well as traffic patterns of simulated hosts. Additionally – since in real networks malicious nodes and nodes that temporarily misbehave exist – it should be possible to define existence and characteristics of simulated attack traffic, too. The creation of realistic and complex simulation environments should be easy and repeatable. Furthermore, this process should be done automatically and include each aspect of simulation environments into one tool in order to save time and avoid errors. This also ensures that the results of different research activities can be compared with each other due to the same simulation premises.

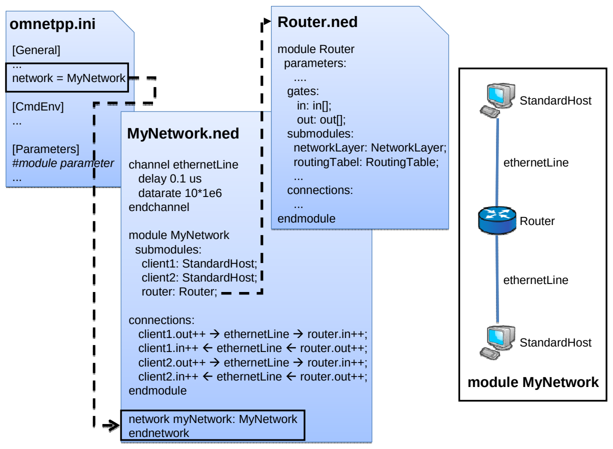
There are existing tools like BRITE [11], Inet [21], or DDoSSim [9] as well as different traffic generators that can be used in order to generate simulation environments. Inet and BRITE, however, are only able to generate topologies. DDoSSim primarily details generation of DDoS attack traffic. Generation of normal traffic patterns is not explained. Traffic generators in most cases focus only on generation of normal traffic patterns but do not regard attack traffic or topology creation. In summary, most tools do not consider all previously mentioned aspects of a simulation environment. Furthermore, most available tools are not able to generate a realistic simulation environment since they apply outdated assumptions or too much simplifications. Therefore, we present in this paper *ReaSE* – a tool that creates simulation environments as realistic as possible based on current state of the art solutions.

The rest of this paper is structured as follows: section 2 details how simulation environments for OMNeT++ are set up in case of Internet simulations. Section 3 then describes the applied methods for generation of topologies, traffic patterns, and attack traffic. Additionally, implementation details are given and we verify that the resulting environments really show the intended characteristics. Related work is described in the respective subsections. Finally, section 4 gives a conclusion and outlook to future work.

2. HOW TO SET UP AN OMNET++ SIMULATION

A simulation in OMNeT++ is realized based on hierarchically structured *modules* that contain the simulation's main functionality. These modules then are executed by the OMNeT++ simulation kernel. So called *simple modules* combine one or more C++ classes that realize the actual functionality, e.g. the implementation of the IP protocol. A *compound module* enables the aggregation of multiple simple modules and therefore, defines the contained simple modules as well as their interconnections with each other. Thus, it is possible to define a compound module that realizes the complete functionality of a standard host system or a router. Compound modules themselves then can be interconnected with other modules by incoming and outgoing gates that specify a so called *channel*. Such a channel, in turn, can possess a certain bandwidth and packet delay.

At least two files are necessary for configuration of an OMNeT++ simulation: the file *omnetpp.ini* specifies some global parameters, e.g. which module actually starts the simulation. *NED* files (*.ned), on the other hand, describe the simulated network and the modules used. Each module defines its gates, parameters, and submodules in a separate *NED* file. A global *NED* file finally specifies the interconnection of these modules that form the simulated network.



omnetpp.ini
```
[General]
...
network = MyNetwork

[CmdEnv]
...

[Parameters]
#module parameter
...
```

MyNetwork.ned
```
channel ethernetLine
  delay 0.1 us
  datarate 10*1e6
endchannel

module MyNetwork
 submodules:
  client1: StandardHost;
  client2: StandardHost;
  router: Router;

connections:
  client1.out++ → ethernetLine → router.in++;
  client1.in++ ← ethernetLine ← router.out++;
  client2.out++ → ethernetLine → router.in++;
  client2.in++ ← ethernetLine ← router.out++;
endmodule

network myNetwork: MyNetwork
endnetwork
```

Router.ned
```
module Router
  parameters:
    ....
   gates:
     in: in[];
     out: out[];
   submodules:
    networkLayer: NetworkLayer;
    routingTabel: RoutingTable;
    ...
   connections:
    ...
endmodule
```

Figure 1: An exemplary simulation setup for OMNeT++

Figure 1 shows an example of the files necessary for an OMNeT++ simulation. The exemplary scenario consists of three nodes: one router and two standard hosts that are interconnected by channels. The file *omnetpp.ini* specifies the starting module `MyNetwork`. This compound module is defined in the *NED* file *MyNetwork.ned* and consists of a channel definition, contained submodules and interconnections between these submodules. The submodules themselves are defined in separate *NED* files. The compound module `Router`, for example, consists of the modules `NetworkLayer` and `RoutingTable` as well as some gates and interconnections of these submodules.

A very popular simulation model for OMNeT++ is the INET framework [1]. It is used to simulate Internet-specific networks that use the MAC, IP, and transport layer of the TCP/IP stack and therefore, it implements these layers for usage within OMNeT++ into simple modules. Furthermore, it provides the compound modules `StandardHost` and `Router` which aggregate functionalities of end and intermediate systems, respectively.

3. SIMULATION ENVIRONMENT

In the following the three aspects of simulation environments identified in section 1 are described in more detail. First the generation of realistic topologies is detailed. This aspect in turn is divided into two parts due to the hierarchical structure of the Internet. The AS level topology, on the one hand, (see section 3.1) focuses on the connection of multiple separate administrative domains, the Autonomous Systems (AS). On the other hand, the router level topology of each AS (see section 3.2) has to be generated, too. The further aspects are generation of realistic traffic patterns consisting of multiple client/server-based traffic flows (see section 3.3) and attack traffic caused by DDoS attacks as well as worm propagations (see section 3.4).

3.1 AS Level Topology

In order to generate realistic AS level topologies there are two possible approaches: based on real observations, e.g. BGP routing data, or using a random topology generator. The first approach generates a very realistic topology based on a single snapshot that is based on real observations at a certain time, e.g. collected by the Routeviews project [12]. A drawback of this approach is that it is difficult to get all the necessary data and to evaluate the huge amount of data for only one simulation environment. Another problem is imposed by the fact that real topologies and BGP routing data are changing now and then and thus, simulation environments must be recreated repeatedly. Thus, we decided to make use of the topology generation method, since in this way it is possible to easily create multiple realistic topologies for evaluation.

Currently, the research community relies on random topologies whose graphs show a power-law distribution in node degree. This means that most nodes have only few edges whereas few nodes have lots of edges. The existing topology generators BRITE and Inet are based on these assumptions, too. This means that both BRITE and Inet are able to create AS level topologies that show a power-law distribution in node degree and thus, highly resemble real networks. BRITE, however, is not maintained any more but can be used for creation of AS level topologies in combination with the BRITE plugin [18] that exports topologies into the *NED* file format of OMNeT++. Inet may be used for creation of AS level topologies but does not offer a possibility to generate according router level topologies afterwards.

Therefore, we implemented a new topology generator into

ReaSE that is based on the positive-feedback preference model (PFP) [24]. This is an iterative growth model that creates topologies with power-law distributed node degree randomly. It starts with three meshed nodes and iteratively adds another node and two or three edges randomly until a given number of nodes is reached. The correctness of this algorithm is proved by the authors. Furthermore, the *rich club* feature [23], i.e. nodes with high degree are frequently connected with other high-degree nodes, is considered in this algorithm, too. After topology creation we, finally, classify each Autonomous System as either *stub AS* or *transit AS* in a way that complete reachability of each AS is ensured by crossing transit systems only.

3.1.1 Implementation and Evaluation

The process of AS topology generation takes an XML-based configuration file as input. Based on given parameters like the number of Autonomous Systems to generate or values for some parameters of the PFP model *ReaSE* creates a single *NED* file. This defines the required number of Autonomous Systems and their interconnections according to the topology generated by the PFP model. Each transit and stub AS is included into the compound module `Internet`, which actually starts the simulation. Additionally, each AS may contain its own router level topology. The *channels* between different Autonomous Systems are assigned a constant bandwidth that may differ between transit/transit, stub/transit, and stub/stub interconnections. Two transit AS, for example, currently are connected with a bandwidth of 10 Gbit/s. Furthermore, the delay of each channel gets a fixed value of 1 ms. This means that the delay with *ReaSE* currently does not depend on a node's geographic position.

In order to verify correctness of our implementation we created multiple AS level topologies consisting of 10 000 Autonomous Systems and compared the three power-law values [8] of the resulting topologies with values of some reference topologies [16]. This comparison resulted in a correlation value of 0.99 and thus, shows that our AS level topology generation works correctly.

3.2 Router Level Topology

On router level the generation of realistic topologies based on real observations is even more difficult than on AS level due to the concerns of commercial ISPs that publishing their topology data will reveal information about their customers. Thus, on router level we again decided to use random topology generators in order to get multiple and different realistic topologies for evaluation. The evolution in case of router level topologies ranges from random graph models [19] over approaches that guarantee certain non-random design principles that are in common use [22] to the already mentioned power-law distribution in node degree. BRITE applies the latter approach, Inet does not generate a router level topology at all.

In [10] the authors claim that these approaches are outdated and they propose a heuristically optimal topology (HOT) approach for generating realistic router level topologies. Their approach is based on the assumption that topology creation within an AS must regard not only power-law distributions but market demands, link costs and hardware constraints, too. This results in a hierarchical topology consisting of few meshed *core* nodes with low node degree that forward aggregated traffic of a high number of *gateway* nodes with high node degree. *Edge* nodes with a node degree of 1 that connect host systems to the Internet complete the hierarchical topology (see figure 2). In summary link bandwidth increases from edge to core whereas connectivity decreases. BRITE, however, does not regard these additional requirements imposed by real networks but only generates random topologies with power-law distribution in node degree.

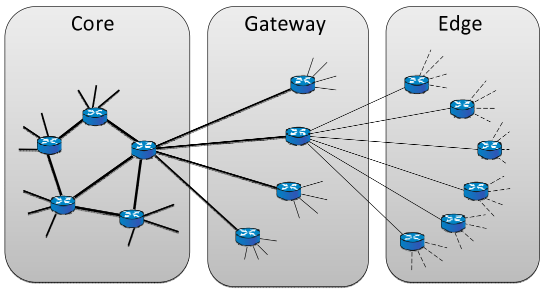


Figure 2: Hierarchy of router level topology

3.2.1 Implementation

Based on the *NED* file created during AS topology generation, in a second step each AS is filled with independently created HOT router level topologies of varying sizes. Therefore, each node of the router level topology is realized either by the module `Router` or `StandardHost` of the INET framework (see section 2). The differentiation between different node types, e.g. core and gateway routers, is achieved by assigning different link bandwidths, e.g. to core/core or core/gateway channels. Between core routers, for example, currently a bandwidth of 2.5 Gbit/s is used whereas only 768 kbit/s are used between host systems and edge routers. The values that actually are used during creation of router level topologies, e.g. the number of routers per topology or the different link bandwidth values, can be specified in a XML-based configuration file. Thus, arbitrary realistic topologies can be created for OMNeT++ simulations easily.

3.3 Traffic Generation

Having created a suitable topology realistic traffic patterns between simulated hosts have to be generated in order to get meaningful evaluation results. Realistic in this case means that the generated traffic shows self-similar behavior [4] and is based on a reasonable mixture of different kinds of traffic. Multiple traffic generators currently exist, e.g. BonnTraffic [15], TrafGen [5], or D-ITG [3]. BonnTraffic is a simulator-independent traffic generator that needs an exporter for each specified simulator. An exporter for NS-2 is already included, for OMNeT++ no exporter seems to exist currently. TrafGen focuses on traffic generation for OMNeT++. Both tools, however, define the parameters of certain traffic flows between a client and a server. Since the endpoints of each flow have to be configured manually there is a huge configuration overhead for simulations including thousands of nodes. D-ITG is an example for a traffic generator that creates real traffic on real systems and is not built for use with simulators.

One possibility to achieve self-similar traffic behavior is to superpose multiple traffic sources that are switched on and off based on heavy-tailed intervals [20]. Another possibility is to use heavy-tailed packet sizes for different traffic flows [14]. In our implementation we used both mechanisms heavy-tailed ON/OFF intervals as well as heavy-tailed packet sizes to ensure self-similar behavior. In order to ensure that a reasonable mixture of different protocols is used during simulations we defined – according to [13] – eight different *traffic profiles* which are based on varying transport protocols: TCP, UDP, and ICMP. Table 1 shows these profiles as well as their according transport protocols and selection probabilities.

Table 1: Traffic profiles currently used with ReaSE

Traffic profile label	Transport protocol	Selection probability
Backup traffic	TCP	1.57
Interactive traffic	TCP	4.71
Web traffic	TCP	11.52
Mail traffic	TCP	4.19
Nameserver traffic	UDP	56.54
Streaming traffic	UDP	1.05
Misc traffic	UDP	14.14
Ping traffic	ICMP	6.28

All host systems defined by the router level topology are divided into two categories before starting a simulation: client and server systems. Client systems repeatedly start a new *traffic flow* consisting of multiple ON/OFF intervals by randomly selecting one of the available traffic profiles depending on their selection probability (see table 1). Then, the traffic profile randomly decides if this traffic flow is destined for a server system within the client's AS or beyond the AS boundaries. Afterwards, the client actively sends traffic according to the traffic profile to the selected server. The parameters for request and reply packet lengths as well as ON/OFF intervals – given in a XML-based configuration file (see figure 3 for an exemplary cutout) – serve as initialization values for a pareto distribution, i.e., a certain heavy-tailed distribution. Servers, in contrast to clients, are passive entities that just answer the requests of clients according to their specific role.

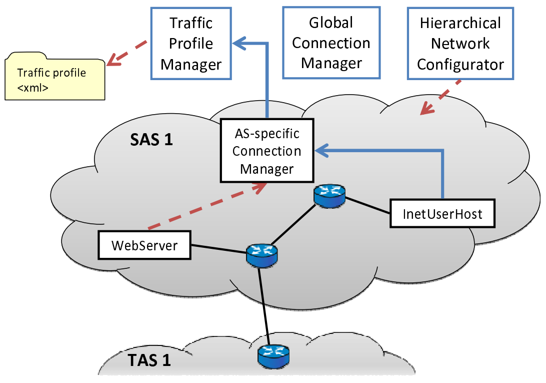
```
<Profile>
        <Id>0</Id>
        <Label>Backup Traffic</Label>
        <RequestLength>10000</RequestLength>
        <RequestsPerFlow>100</RequestsPerFlow>
        <ReplyLength>100</ReplyLength>
        <ReplyPerRequest>1</ReplyPerRequest>
        <TimeBetweenRequests>0.01</TimeBetweenRequests>
        <TimeToRespond>0.1</TimeToRespond>
        <TimeBetweenFlow>5.0</TimeBetweenFlow>
        <SelectionProbability>15</SelectionProbability>
        <WANProbability>33</WANProbability>
</Profile>
```

Figure 3: Exemplary traffic profile – Backup traffic

Thus, by using such traffic profiles we provide an easily configurable and extensible way of generating traffic patterns while additionally ensuring that the resulting aggregated traffic shows self-similar behavior.

3.3.1 Implementation and Evaluation

In a first step the differentiation of the router level topologies' host systems into clients and servers has to be realized. This is achieved by redefining all `StandardHosts` of the *NED* file, which contains the topology, according to their new role. Clients are represented by the module `InetUserHost`, servers e.g. by the modules `WebServer` or `NameServer`. Different server roles identify the support for certain traffic profiles and transport protocols. A `WebServer`, for example, is only able to respond to requests of the traffic profile `Web traffic`, which is based on TCP (see table 1).



Traffic profile <xml>
Traffic Profile Manager
Global Connection Manager
Hierarchical Network Configurator
SAS 1
AS-specific Connection Manager
InetUserHost
WebServer
TAS 1

Figure 4: Small exemplary simulation environment

In order to support generation of realistic traffic patterns we had to extend the INET framework of OMNeT++ by additional modules: the `HierarchicalNetworkConfigurator`, the `TrafficProfileManager`, and a global as well as local `ConnectionManagers` per AS. Figure 4 shows a very small simulation environment containing all these new modules. Additionally, the given stub AS `SAS 1` contains two host systems – one client and one server – as well as two routers.

At simulation startup the simple module `Hierarchical-NetworkConfigurator` assigns unique IP addresses to all nodes of the topology. Currently, *ReaSE* assigns a \16 prefix to each Autonomous System of the AS level topology, i.e. each AS may contain up to 65 536 router level nodes. Then, the `HierarchicalNetworkConfigurator` creates static routes within each AS as well as between different Autonomous Systems based on shortest paths between source and destination. On AS level the role of each AS is additionally regarded. This means that only transit AS forward traffic. A stub AS, on the other hand, may only appear at the beginning or at the end of a routing path. Afterwards, the simple module `TrafficProfileManager` reads all available traffic profiles from a XML-based configuration file. Finally, every server module has to register its IP address and its role with the local `ConnectionManager` of the AS the node belongs to. The local `ConnectionManagers` in turn register at the global `ConnectionManager`. All these actions that have to be performed before the simulation actually can start are marked with red dashed arrows in figures 4 and 5.

Figure 5 shows the procedure of a client starting a new traffic flow. Having completed simulation startup the client – represented by the module `InetUserHost` – has to select a traffic profile for its first traffic flow. Therefore, it asks the AS-specific `ConnectionManager` for a server address and a traffic profile using the method `getServer`. The `Con-`

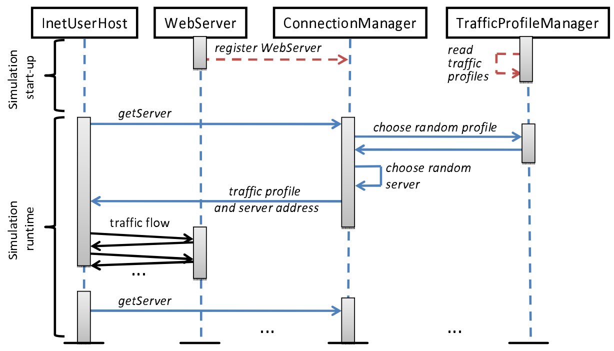


Figure 5: Procedure of traffic profile selection

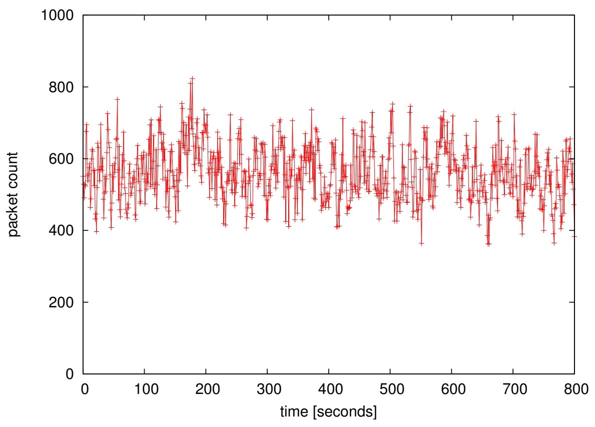


Figure 6: Traffic at gateway router

`nectionManager` then requests a traffic profile at the `TrafficProfileManager`, which manages all available traffic profiles. This module randomly selects a traffic profile based on the given selection probabilities. Additionally, it decides if the traffic flow takes place between the client and a server within the client's AS or beyond the AS boundaries. This decision is taken based on the profile's WAN selection probability (see figure 3). Finally, the `TrafficProfileManager` transmits the selected traffic profile to the AS-specific `ConnectionManager`. In our example the traffic profile `Web traffic` was randomly selected and the communication takes place between the client and a web server within the client's AS.

If the traffic flow takes place with a server within the client's AS the `ConnectionManager` randomly selects one of the registered servers and communicates the selected traffic profile and server address to the client. Afterwards, the traffic flow is started based on the parameters of this traffic profile. If the communication, however, takes place with a server in another AS the AS-specific `ConnectionManager` has to request a server address at the global `ConnectionManager`. This module then forwards the request to an arbitrary AS-specific `ConnectionManager` that selects an appropriate server within its AS. Finally, the server address is transmitted to the client's AS-specific `ConnectionManager` that, in turn, communicates this address as well as the selected traffic profile to the client. As soon as the traffic flow is completed the selection procedure is started again.

In order to verify the existence of self-similarity in the resulting traffic we conducted multiple OMNeT++ simulations based on a topology with 90 000 hosts within 30 Autonomous Systems using 8 traffic profiles. We observed the packet counts of about 1 000 randomly selected routers – 900 edges, 80 gateways, and 30 core routers – and calculated the Hurst value based on the method of m-aggregated variances for different interval durations (see table 2). Figure 6 additionally shows traffic, which looks self-similar, that is recorded at a randomly selected gateway router. In table 2 we can see that the Hurst value is significantly greater than 0.5 in nearly all cases. This means that the traffic generated with *ReaSE* really shows self-similar behavior.

3.4 Addition of Attack Traffic

If real networks should be simulated traffic patterns as previously described are not enough since these are based on the assumption that all nodes behave correctly. Furthermore, if systems for attack prevention, detection, or reaction are evaluated not only misbehaving but also malicious nodes are needed. Therefore, we extended our traffic generation by a further aspect: realistic attack traffic. DDoSSim has the same intention but neither a detailed description nor the implementation itself is freely available. Thus, we decided to extend the INET framework of OMNeT++ by some further modules that enable the simulation of genuine DDoS attacks and worm propagations.

Table 2: Hurst value for different node types

node type	interval	average	std dev
Edge	100ms	0.7414	0.095
Edge	1s	0.7128	0.079
Gateway	100ms	0,8934	0.0814
Gateway	1s	0.8645	0.0675
Core	100ms	0.9610	0.0593
Core	1s	0.9475	0.0480

3.4.1 DDoS Attacks

In order to simulate distributed denial-of-service (DDoS) attacks we integrated a real tool for conducting such attacks: the Tribe Flood Network [6]. By integrating this DDoS tool our attack traffic reproduces the specific characteristics of DDoS attacks, e.g. ramp-up behavior at the beginning of an attack or multiple attack waves. Each attacking system – a so called *zombie* – can, however, only take part in a single attack at the same time. In order to generate such DDoS attack traffic certain configuration parameters, e.g. the number of attacking systems, the number of attack waves, the starting times of the attacks, and the victim address, must be given. These parameters are specified in the file *omnetpp.ini*.

Before being able to actually start a simulation the zombie systems have to be distributed within the already created network specified in the *NED* file. *ReaSE* performs a random distribution by replacing randomly selected client systems with zombie systems. This is realized by redefining these randomly selected clients from `InetUserHost` to `DDoSZombie` within the existing *NED* file. The module `DDoSZom-`

`bie` is realized as compound module that contains the simple module `TribeFloodNetwork` as well as some other modules provided by the INET framework that are necessary to accomplish the complete functionality of an attacking system, e.g. a network layer and a routing table. The simple module `TribeFloodNetwork` implements the actual functionality of creating attack packets according to the configuration parameters. These packets then are sent directly on IP layer of the INET framework. Thus, the raw socket that is used by the real attack tool is replicated.

DDoS attacks in most cases are based on the fact that some resources like memory or link bandwidth at a victim system can be overloaded by the attack traffic and then, certain services offered by the victim are not available any more. This means, that a host system may only support a limited number of TCP connections or that a router must have limited packet queues in order to achieve overload situations caused by flooding attacks like DDoS. Most network simulators including OMNeT++, however, do not provide the possibility to simulate such overload situations of simulated nodes. Therefore, we had to do some changes in the INET framework of OMNeT++. First, we modified the INET TCP implementation in a way that only a limited number of TCP connections is supported. Additionally, we had to change the TCP state machine in a way that a single TCP SYN packet does not create connection state that is stored till the end of the simulation but only for a certain time if no further TCP packet for this connection is received. This ensures that new connections can be accepted after a certain time period.

Figure 7 shows traffic at the victim host of a TCP SYN flooding attack [7]. It is obvious that the number of incoming TCP SYN packets is not limited but the number of outgoing TCP SYN/ACK packets – and thus, the number of connection states that can be stored at a host system – actually is limited, in this example to 250. This causes an overload situation at the victim. Another example for overload situations are limited packet queues of routers that lead to packet drops during overload.

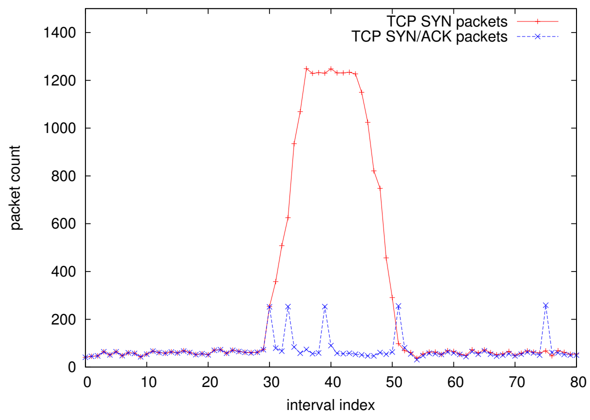


Figure 7: Traffic at a DDoS victim host

3.4.2 *Worm propagations*

In case of worm propagations we implemented two different alternatives: the first one is based on UDP, the latter on TCP. Both alternatives are based on a rather simple probing mechanism that was, for example, used with Code Red I [25]. Systems that are infected by a worm start sending probing packets to other randomly selected hosts. If a vulnerable host that currently is not infected receives such a probing packet this host changes its state to infected and starts sending probing packets to randomly selected hosts itself. Hosts that are not vulnerable, e.g. because the do not offer the service that should be infected, respond with an ICMP port unreachable message to such probing packets and then, silently discard them. If a probing packet is sent to an IP address that is not assigned to any host an ICMP destination unreachable message is created by a router. The creation of ICMP error messages is provided by the INET framework.

Before starting a simulation it must be specified which nodes are infected at simulation startup and which nodes are vulnerable by probing packets. Therefore, randomly selected clients represented by the module `InetUserHost` in the existing *NED* file are redefined as `UDPWormVictim` and `TCPWormVictim`, respectively. The configuration of worm parameters, e.g. probing frequency, destination port, or IP address range for probing packets, again is done in the *omnetpp.ini* file. The nodes that are infected at beginning of the simulation are randomly selected based on an infection probability, which is specified in the configuration. If, for example, 20 % of the worm nodes should be infected each node calculates a random number and decides if it is infected or just vulnerable based on this number.

4. CONCLUSION AND OUTLOOK

In this paper we presented *ReaSE*, a tool for creation of realistic simulation environments for OMNeT++ that considers not only a single aspect of such an environment but three important aspects: topology generation on both AS and router level, traffic patterns, and attack traffic. The creation of such a simulation environment is based on current state of the art solutions. Additionally, we verified that our tool really creates simulation environments showing the intended characteristics. The tool currently is in an early development state but should be published as soon as possible on *http://www.tm.uka.de/∼gamer*.

Currently, creation of realistic simulation environments is based on XML configuration files but we are working on a graphical user interface that should further simplifie this task by hiding implementation details from users. Additionally, we aim at further extending two aspects of *ReaSE*: more traffic profiles should be integrated, e.g. based on application layer traffic. Furthermore, currently only DDoS attack traffic and worm propagations, which are based on a very simple probing mechanism, can be generated. Thus, further attack classes should be added in the future. Finally, we did not consider routing mechanisms in this work but only used static, single-path routing. Thus, other mechanisms should be taken into account, too. Additionally, the currently fixed delay of channels could be calculated depending on a geographic position of nodes.

5. REFERENCES

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

[2] Mobility Framework. http://mobility-fw.sourceforge.net/, Jan. 2007.

[3] S. Avallone, D. Emma, A. Pescap, and G. Ventre. A Practical Demonstration of Network Traffic Generation. In *Proc. of the 8th IMSA*, pages 138–143, Aug. 2004.

[4] 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.

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

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

[7] W. M. Eddy. Defenses Against TCP SYN Flooding Attacks. *Cisco Internet Protocol Journal*, 8(4), Dec. 2006.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. *Computer Communication Review*, 29(4):251–262, 1999.

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

[10] 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.

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

[12] University of Oregon. Route Views Project. http://www.routeviews.org.

[13] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proc. of the Internet Measurement Conference 2005*, pages 15–28. USENIX Association, Oct. 2005.

[14] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. of International Conference on Network Protocols*, pages 171–180, Oct. 1996.

[15] B. Roemer. BonnTraffic: A modular framework for generating synthetic traffic for network simulations, Nov. 2005. http://web.informatik.uni-bonn.de/IV/bomonet/BonnTraffic.htm.

[16] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos. Power laws and the AS-level Internet Topology. *IEEE/ACM Transactions on Networking*, 11(4):514–524, Aug. 2003.

[17] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of European Simulation Multiconference*, June 2001.

[18] A. Varga. OMNeT++ export for BRITE 2.1. http://www.omnetpp.org/filemgmt/singlefile.php?lid=5, 2003.

[19] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.

[20] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level. In *Proc. of ACM SIGCOMM*, pages 100–113, Sept. 1995.

[21] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report UM-CSE-TR-456-02, University of Michigan, July 2002.

[22] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. *Proc. of IEEE INFOCOM*, 2:594–602, Mar. 1996.

[23] S. Zhou and R. J. Mondragon. The Rich-Club Phenomenon In The Internet Topology. *IEEE Communications Letters*, 8(3):180–182, Mar. 2004.

[24] 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.

[25] 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.