Chapter 5

Design and implementation

5.1 Resources and tools

5.1.1 Hardware

Simulation hardware

The research institute Fraunhofer Fokus provides a machine to intensively run simulations. It is a Dual Quad-Core Intel Low Volt Xeon L5310 processors at 1.6GHz (Dell poweredge 1950) with 4 GB RAM and GNU Linux Ubuntu Server distribution with SMP activated.

The simulation machine hardware and operative system configuration permits distributed simulations through shared memory as well as running different instances of the simulator at the same time. During the development of this thesis magpie was available only for researchers involved in this thesis.

Development hardware

Another machine is provided by the institute as a work station for implementing the simulator, testing and debugging, parsing datasets and results, and other activities related with the topic of this thesis. In particular, this machine is a Pentium 4 at 2.0 Ghz and 512 MB of RAM memory. The Gnu Linux Xubuntu distribution was installed to perform the topics listed above.

5.1.2 Simulators

Table 5.1 shows a list of commonly used network simulators. We list some characteristics in order to do a coarse grained selection of which simulators might be appropriate for our topic. We list the scalability of the simulator, how topologies are organized, if framework allows simulation, emulation or both, organizations supporting the framework, type of license and development state. The topology attribute notes how topologies are organized and packets are routed. Namely, at which layer packets are transferred and how nodes are identified.

Simulation tools coarse grained selection

We do not consider tools that do not permit parallel and distributed simulations in case we decide to develop such type of simulations with the simulation machine. Therefore, not scalable tools (i.e. GloMoSiM,J-SIM) are not considered. We also do not intend to use those tools whose code is not distributed but we accept those tools which license allows commercial use and are able to be modified. Namely, we choose tools that allow code modification, as this is requirement of our topic. In addition, we do not consider those tools which are obsolete, unmaintained or not enough mature (i.e
Table 5.1: Most common network simulators

<table>
<thead>
<tr>
<th>Tool</th>
<th>Scalability</th>
<th>Topology</th>
<th>Simulate</th>
<th>Emulate</th>
<th>Support</th>
<th>Develop.</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS2</td>
<td>single</td>
<td>routing node id</td>
<td>yes</td>
<td>yes</td>
<td>DARPA, Xerox</td>
<td>Stable</td>
<td>Public</td>
</tr>
<tr>
<td>NS3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Alpha</td>
<td>Public</td>
</tr>
<tr>
<td>PDNS</td>
<td>parallel</td>
<td>node id</td>
<td>yes</td>
<td>yes</td>
<td>GIT</td>
<td>Obsolete</td>
<td>Public</td>
</tr>
<tr>
<td>PARSEC</td>
<td>single</td>
<td>wireless</td>
<td>yes</td>
<td>no</td>
<td>Glomosim</td>
<td>Developing</td>
<td>Public</td>
</tr>
<tr>
<td>DaSSF</td>
<td>parallel/</td>
<td></td>
<td>yes</td>
<td>no</td>
<td>CSM</td>
<td>Unmaint.</td>
<td>Public</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrimeSSF</td>
<td>parallel/</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>CSM</td>
<td>Stable</td>
<td>Public</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTNetS</td>
<td>parallel/</td>
<td>TCP/IP</td>
<td>yes</td>
<td>no</td>
<td>GIT</td>
<td>Stable</td>
<td>Public</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNet</td>
<td>no</td>
<td>routing data link</td>
<td>yes</td>
<td>no</td>
<td>Western Australia</td>
<td>Stable</td>
<td>public</td>
</tr>
<tr>
<td>Simmcast</td>
<td>single</td>
<td>Abstraction</td>
<td>yes</td>
<td>no</td>
<td>Uni.Vale do Rio dos Sinos</td>
<td>Unmaint?</td>
<td>public</td>
</tr>
<tr>
<td></td>
<td></td>
<td>simulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSim</td>
<td>NA</td>
<td>components</td>
<td>yes</td>
<td>no</td>
<td>Ohio University</td>
<td>Unmaint?</td>
<td>public</td>
</tr>
<tr>
<td>Hegons</td>
<td>NA</td>
<td>physical</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>stable</td>
<td>Free non commer.</td>
</tr>
<tr>
<td>Swans</td>
<td>NA</td>
<td>wireless</td>
<td>yes</td>
<td>no</td>
<td>Conrell</td>
<td>stable</td>
<td>Free non commer.</td>
</tr>
<tr>
<td>NetWiser</td>
<td>NA</td>
<td></td>
<td>yes</td>
<td>no</td>
<td>Bitwiser</td>
<td>Beta</td>
<td>Commer./Public</td>
</tr>
<tr>
<td>OMNeT++</td>
<td>parallel</td>
<td>device oriented</td>
<td>yes</td>
<td>no</td>
<td>OMNET</td>
<td>Stable</td>
<td>Commer./Public</td>
</tr>
<tr>
<td>NCTUns</td>
<td>NA</td>
<td>Devices Protocols</td>
<td>yes</td>
<td>no</td>
<td>Simreal</td>
<td>Stable?</td>
<td>Commer.</td>
</tr>
<tr>
<td>OPNET</td>
<td>parallel</td>
<td>device oriented</td>
<td>yes</td>
<td>no</td>
<td>OPNET</td>
<td>Stable</td>
<td>Commer.</td>
</tr>
<tr>
<td></td>
<td>/distributed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shunra</td>
<td>single</td>
<td>device oriented</td>
<td>yes</td>
<td>no</td>
<td>SHUNRA</td>
<td>Stable</td>
<td>Commerc.</td>
</tr>
<tr>
<td>Netscale</td>
<td>single?</td>
<td>routing link layer</td>
<td>yes</td>
<td>no</td>
<td>N2NSoft</td>
<td>Stable?</td>
<td>Comm.</td>
</tr>
</tbody>
</table>
5.1. RESOURCES AND TOOLS

NS3, DaSSF). Tools which rely on a heavily tightness to device definition are discarded (i.e. OPNET, OMNET++) because our topic is not directly related to device performance and analysis. After applying this criteria we selected NS-2, PDNS, PRIME and GTNETS as the more suitable simulation tools and are listed in Table 5.2

<table>
<thead>
<tr>
<th>Tool</th>
<th>Goal</th>
<th>License</th>
<th>Language</th>
<th>Support</th>
<th>Last Release</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS-2</td>
<td>simulation of TCP, routing, and multicast protocols over wired and</td>
<td>GPL?</td>
<td>C++ OTCI</td>
<td>Xerox, Darpa SCU</td>
<td>2.31</td>
<td>Mar. 2007</td>
</tr>
<tr>
<td></td>
<td>wireless (local and satellite) networks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDNS</td>
<td>extensions and enhancements to the ns simulator to allow a network</td>
<td>GPL</td>
<td>C++ OTCI</td>
<td>GIT (Dr. George Riley)</td>
<td>ns-2.27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>simulation to be run in a parallel and distributed fashion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIME (SSF)</td>
<td>Parallel Real-time Immersive network Modeling Environment</td>
<td>GPL</td>
<td>C++ DML</td>
<td>National Science</td>
<td>Prime1.0.1</td>
<td>Nov. 2006</td>
</tr>
<tr>
<td></td>
<td>(PRIME) SSFNet is a high-performance network simulator for: Deal</td>
<td></td>
<td></td>
<td>Foundation CAREER Award CSM</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>with large-scale real-time interactive network simulations and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>models for background network traffic modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTNETS</td>
<td>Study the behaviour of moderate to large scale networks. Create a</td>
<td>GPL (?)</td>
<td>C++</td>
<td>GIT (Dr. George Riley)</td>
<td>1.3</td>
<td>Aug. 2006</td>
</tr>
<tr>
<td></td>
<td>simulation environment that is structured much like actual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>networks are structured. For example, in GTNetS, there is clear and distinct separation of protocol stack layers.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Simulation tools fine grained selection

PDNS extension of NS-2 was developed by Dr. George Riley (GIT). Afterwards he implemented GTNetS simulator. Dr. Riley is currently focusing on the development of NS-3. One of the goals of NS-3 is to maintain compatibility with NS-2 and GTNetS. Unfortunately NS-3 is still at alpha stage. SSFNet (DaSSF version) is an unmaintained simulator, whose main developer, Jason Liu, moved towards developing the PRIME SSF based on DaSSF. PRIME SSF has novel approaches, as using GPU units, application emulation and flow traffic between others. However no relevant simulations have been done with it at the time of this writing (June 2007).

We discuss below these simulation tools characteristics in order to choose which one applies to our study. We therefore focus on:

i. Performance

ii. Scalability
iii. Usability

iv. Related work

In terms of performance, both NS-2 and GTNetS have been found to have similar performance in a single machine [WeMS05] or in a parallel/distributed environment [FPPW03]. However, the authors in [PerS04] state that PDNS is fast for long-lived / high throughput connections and GTNetS fares better for mice connections (short-lived).

The authors in [Nico02] perform a comparison between NS2, SSFNet and others. Authors found that NS-2 requires more memory than SSF but is slightly faster than SSF. NS-2 authors argue that such memory requirements are devoted to TCL integration, which requires more memory but provides advantages to users and developers.

In terms of scalability, the authors in [FPPW03] present a study comparing PDNS and GTNetS and proof their capability to simulate large topologies of millions of nodes. The authors in [Rile03] state that inexpensive workstations with 2GB of main memory can achieve a topology size of 177,000 nodes with GTNetS.

In terms of usability and topology definition, we remark that GTNetS focus on connection oriented applications and connectionless applications. Namely, network nodes are defined with IP addresses and simulation focus on the application layer. In the other hand NS2 nodes do not have IP address as this simulator was design for routing simulations. Furthermore, GTNetS is build with C++, while PDNS uses C and a TCL. Therefore, we assume the learning curve of PDNS to be slower than GTNetS. SSF has implementations in two different languages: Java and C++.

Prior to deciding which simulator to use, we check what simulator is most usually used by the research community relating to our topic: worms and DNS. We found that NS2 was used in most of the papers using network simulation. We found related work for worms in all three simulators but DNS related work is only apparent for NS-2. However, no email worm related work is apparent for such simulators.

We decide to use GTNetS as framework because it has the best trade-off of requirements. Namely, it performs better for short lived connections than PDNS. We remind the reader that DNS uses the UDP protocol for most of its operations, which is short lived. In addition, GTNetS is highly scalable, and we choose it to allow distributed and parallel simulations. GTNetS is build with C++. Thus, no need of additional languages is needed. Related work relating scanning worms is apparent for GTNetS and accepted. In addition GTNetS is used to build NS-3, which might be the most used simulator as NS-2 is nowadays. Finally, in GTNetS topologies are organized with nodes having IP addresses, which facilitates modeling applications as they are in the real world. In contrast, NS-2 and PDNS use identifiers which difficults to model components at the application layer such worms, name servers, SMTP servers and email users.

5.1.3 Graph generators

Our approach specifies generating topologies for the Internet topology at AS level and email networks. Since [FaFF99] two approaches are apparent to generate topologies. Generators might be degree-based or structural based and briefly explained in previous sections. In this section we list several generators already used in related work.

Degree-based generators

Table 5.3 is partially extracted from [ChaF06] and lists several graph generators which might be used for network simulations or social network modeling. We list the most relevant of them but remark the diversity of models apparent in related work as reported per the authors in [ChaF06]. We do not
discuss these model because it is out of the scope of this thesis but we list the characteristics that apply to our work.

Table 5.3: Models and methodologies for generating different graphs

<table>
<thead>
<tr>
<th>Generator</th>
<th>Type</th>
<th>Power Law Degree</th>
<th>Diameter</th>
<th>Clustering C.</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erdős-Rényi</td>
<td>Random graph</td>
<td>O(logN)</td>
<td>Low</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLRG</td>
<td>Random graph</td>
<td>any</td>
<td>O(logN)</td>
<td>CC</td>
<td>Adds degree distribution to random models</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0 for large N</td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td>Preferential attachment</td>
<td>$\gamma = 3$</td>
<td>O(logN)</td>
<td>$CC \propto N^{-0.75}$</td>
<td></td>
</tr>
<tr>
<td>IA</td>
<td>Preferential attachment</td>
<td>$\gamma \in [2, \infty]$</td>
<td></td>
<td></td>
<td>Probability of new nodes gaining new edges</td>
</tr>
<tr>
<td>AB</td>
<td>Preferential attachment</td>
<td>$\gamma \in [2, \infty]$</td>
<td></td>
<td>Higher than AB, BA, PLRG</td>
<td>Edges rewiring and internal edges</td>
</tr>
<tr>
<td>GLP</td>
<td>Preferential attachment</td>
<td>$\gamma \in [2, \infty]$</td>
<td></td>
<td></td>
<td>Stronger preference to connect to high-degree nodes</td>
</tr>
<tr>
<td>Forest fire</td>
<td>Preferential attachment</td>
<td>$\gamma =$?</td>
<td>Shrinking diameter as N grows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small world</td>
<td>Geographical model</td>
<td>O(N) for small N, O(lnN) for large N</td>
<td>$CC(p) \propto (1-p)^3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inet-3.0</td>
<td>Internet specific</td>
<td></td>
<td></td>
<td></td>
<td>Specific for the AS graph</td>
</tr>
<tr>
<td>BRITE</td>
<td>Geographical and preferential attachment</td>
<td>$\gamma =$?</td>
<td>Low</td>
<td>like in BA</td>
<td></td>
</tr>
</tbody>
</table>
Structural-based generators

Several structural generators are apparent. The authors in [HPSS03] compare the two most relevant structural network topology generators for PoP-level (Point of Presence) topologies. Authors the generators Tiers and GT-ITM and provide range parameters for such generators in order to match two well-know real-world ISP topologies. However, such topologies are small when talking about number of nodes, as every node is a point of presence and might have in reality hundreds of routers. Therefore such work might be useful to model small networks or ISPs at PoP level, but not appropriate to model the AS topology of the Internet. Thus, we discard structural generators to model the AS backbone for this thesis experiment.

Topology generator selection

As noted per the authors in [TGHS02], some researchers claim that a structural topology generators might be more appropriate when modeling topologies with small number of nodes (less than 1000). The same authors find that degree-based generators capture the hierarchical nature of the Internet while matching specific graph patterns. However, as we pretend to provide an scalable simulation environment, we use a degree-based generator to model both AS networks and email networks.

At Table 5.3 we present several degree-based graph generators. BRITE and Inet and has been extensively used by the research community. However BRITE is integrated with GTNetS. As stated per the authors in [ChaF06] BRITE provides a flexible system to generate different topologies but there is little discussion about the parameters to fit for different topologies. The authors in [ChaF06] report that the authors of Inet-3 state that Inet does still do not well represent the Internet in terms of clustering coefficient.

As graph patterns at the Internet topology are continually changing with new measurements as those performed regularly by CAIDA, we prefer to use a more flexible topology generator. The authors in [ZoTG04] suggest, based on dataset observations, that the Internet email network is heavy-tailed distributed. They choose to use GLP to model email networks because it models power laws, which are also heavy-tailed distributed, and the power law degree can be adjusted.

Finally, we decide to use GLP because it has been used to generate both physical [BuTo02] and social network topologies [ZoTG04] has adjustable parameters and has scalable properties.

5.1.4 Mathematical tools

Gnu Scientific Library

Several processes related to our topic are modeled using distributions. In particular, uniform and normal distributions are apparent. Therefore, we need a suit to extract values from such distributions. As GTNetS is written in C++, the required suite should be also accessible through run-time execution. Namely, we search for a C++ library capable of modeling these distributions.

We consider two libraries: Boost and GSL. Boost is a more general set of libraries relating to different topics. In the other hand, GSL provides a numerical library and provides more different implementations and algorithms for the distributions we require. Thus, we choose GSLs.

NetMine

In order to study the graph patterns of the generated topologies we propose using the tools presented [CZBF04]. NetMine is a package for extracting patterns from large graphs, such as diameters. NetMine requires Perl and GnuPlot. For certain analysis Matlab is required but we do not intend to use them as they are out of the scope of this thesis.
5.1.5 Parsers and plotters

Parsers

We plan our simulator to provide large amounts of data regarding the epidemic incident. In addition, we have DNS datasets collected with by the system administrators with the Snoop-Dog packet sniffer. Such data must be parsed before obtaining patterns and results. We propose using Perl to parse and extract the important data from files because it is a very flexible language, which might allow more operations to the data than other languages. Perl is open source and distributed under the Artistic License, which is a less restrictive than the GPL.

Plotters

Results must be presented in a human readable form. Thus, we use GnuPlot as a the plotting tool to use to represent several metrics related to the epidemic study. GnuPlot is distributed under the GPL license and is widely used by the research community. We also considered using RRDtool but we discard it because it is oriented to plotting graphs in run-time and is less flexible than GnuPlot.

5.1.6 Compilers and debuggers

Compilers

As GTNetS is build with C++, a C++ compiler is needed to extend GTNetS functionality and provide executables. We use the GNU compiler collection G++. In addition to the GNU C libraries, GTNetS also requires the QT libraries.

Debuggers

In order to debug the code from run-time errors we use the Gnu Debugger (GDB). Asides GDB, we use other debuggers which might report more information than GDB. Namely, we pretend to use Valgrind in order to detect memory leaks.

5.2 Class diagram and requirements

The class diagram is partially constrained by the GTNetS architecture, detailed at the reference manual [Ripe04]. The GTNetS architecture follow the design of real networks by mimicking its layer structure. Namely, GTNetS allows implementing components at the application layer and layers 2,3 and 4. In addition, GTNetS architecture defines how topologies are built. In particular, topologies are build upon nodes, which are interconnected through links. The programmer can freely combine these components in order to compose different topologies.

Finally, as GTNetS is designed to mimic real-world networks components, it allows to implement these components with the real-world specifications. Namely, the class diagram represents the real world components apparent at a worm epidemic: name server, SMTP server, users and worms. Therefore, we first describe the class diagram because the use cases follow the real world use cases. We broadly explain below the functionality and requirements of each class added to the original code of GTNetS.

5.2.1 Application Layer classes

Name server application class

The name server application implements the Application and TimeHandler interfaces of GTNetS. Namely, the name server application belongs to the application layer of the GTNetS and has schedul-
ing capabilities.

Timing in this application is essential in order to implement our approach. Timing is essential to delay packets given a value obtained from a distribution and activate the sending process at a certain point. Thus, these application must have access to a number generator, which is described later at the StatisticalGenerator class.

The name server application implements query resolving. This means that in order to resolve names, this application requires access to the name tree, which is defined by the DNSTree class. No more DNS operations are defined because they are not involved in our study.

Two hash-map structures are needed to implement the functionality at this class:

i. Hash-map with timestamp representing scheduled time as a key and packet as a value

ii. Hash-map with an IP address as a key and integer representing the number of queries done per machine with IP times as a value

SMTP server application class

The SMTP server application implements the Application interface of GTNetS. This class receives packets and performs different functionality depending on which type of packet arrived. Namely, packets might be either infected emails from worm applications either user queries from user applications. No scheduling is necessary as responses must be processed and send as they arrive.

The SMTP server class must store information such as arrival time of infected emails (for statistical purposes) and number of infected emails at inbox. Therefore a hash-map structures are needed:

i. Hash-map with the id of a social network node as a key and an tuple as a value. The tuple consists of an integer representing the number of infected emails at the inbox and a timestamp of the first infected mail at inbox.

Email user application class

The email user application implements the Application and TimeHandler interfaces of GTNetS. Scheduling capabilities are necessary in order to mimic the behavior of real users, which consists on checking if emails arrived at the SMTP server. Such intervals are obtained through distributions, which must be accessible at the StatisticalGenerator class.

The email user application functionality is simple: queries the SMTP server and if response indicates an infection the email user application attaches the worm application to the physical node the email user is located. Therefore no structures are needed for this application.

Worm application application class

The email worm application implements the Application and TimeHandler interfaces of GTNetS. Scheduling capabilities are necessary to allow scheduling of sending packet events by sequentially accessing a list of victims retrieved from the infected host. The worm functionality is divided in two processes: querying the DNS and sending a packet. The worm retrieves from a list structure the ids of victims and queries the DNS to resolve victims SMTP server. When a response is received it sends to the SMTP an infected copy. Therefore, only one structure is needed:

i. Vector of social network identifiers, representing the nodes with whom infected host social network id is connected to at the social network.
5.2.2 Topology classes

Work station class

The workStation class is a container for data stored at the physical node. These attributes define the IP address of the physical node and its logical name as known by the DNS, the IP address of the the SMTP server that the machine queries for new emails, the IP address of the local name server and the identification number for the AS the node belongs to.

Every physical node is associated to one workStation class allowing accessibility to the data listed below from every class at out simulation. This class is associated to the GTNetS node class, and is the only modification to the original code. We try to decouple our extension from original code for usability and comprehensibility reasons.

Social network class

The social network is a containment class, which other components might query in order to retrieve information relating to the social network. Thus, one structure is needed:

- Hash-multimap, with a social network id as a key and a social network id as value, which means that each pair of ids represent an edge between two social network nodes and there might be replicated keys.

Stub networks classes

We define topologies as abstract classes in order to facilitate introducing new topologies that maintain certain properties. We implement three topologies: small campus, full connected and ring connected graphs. Topologies do not change at run-time and are only necessary to construct new nodes and links. Therefore, no additional structures are needed. In this thesis we implemented three classes: are a Full graph of stars, Ring graph of stars and the Small Campus Network.

5.2.3 Support classes

DNS tree singleton class

The DNS Tree class is an auxiliary class that stores the social network graph in order to be accessed from other simulation components. Namely, class is a data container class for social network ids and IP addresses. This class allows decoupling physical nodes from logical nodes by adding a middle layer. One structure is needed:

- Hash-map with the id of a social network node and the SMTP server it queries to.

Constants static class

The constants class defines user-defined variables. This class should contain all global variables in order to increase the usability and maintainability of the application.

Statistics class

The statistics class gathers statistics form the simulation. However, as user can not interact with the application at execution time due to the scale, rather than keeping them in memory this class writes on-the-fly to the disk the gathered statistics in order to avoid wasting memory.
**Statistical generator class**

The statistical generator class provides functions to retrieve random numbers with a certain distribution. It is implemented with a singleton pattern in order to maintain a good statistical behavior of the generator. Two structures are apparent:

i. Uniform random generator suitable for randomly assigning physical nodes to IP addresses and applications starting time.

ii. Normal distribution number generator needed to model the user behavior and name server responses.

**GLP parser class**

The GLP parser class parses the output file of the GLP graph generator in order to retrieve the a graph for use at the simulation. A containment structure is needed:

i. Hash-multimap, with an integer as a key and another integer as value, where a pair of integers represent an edge between two nodes at the graph.

**5.3 Use cases**

The use-cases present at the simulation reflect real-world processes. However, we prefer here to present the use cases already integrated with the simulation tool GTNetS.

**5.3.1 Email user use cases**

**User queries SMTP**

Email users are scheduled to query their SMTP server in order to check their email. Namely, they receive an event from GTNetS scheduling system in order to activate this use-case. Once the user is activated they prepare a packet with a payload consisting on a token specifying that packet is a query for email and their social-network id. Afterwards application sends to the SMTP server the formed packed and schedules itself with the interval defined at the creation of such user. This use case is presented at Figure 5.1.

![Diagram](image)

Figure 5.1: User time out use case: query the SMTP server for new mails
5.3. USE CASES

User receives SMTP response

Users listen for packets at the port where they sent the query. Once a packet arrives, the packet is checked to contain a valid payload. Response payload contains the number of infected mails at the user mailbox and the time-stamp for them. If user have at least an instance of the email worm then it might be possible to open it. We model the probability that a user opens an infected email as modeled per [ZoTG04]. If user opens email then he gets infected and attaches a worm application to the physical node. This use case is presented at Figure 5.2.

![Figure 5.2: User receives packet use case: receive information about inbox](image)

New email user

New users are created at the start of the simulation. However, they can be created at run-time. The constructor for this class only sets the parameters referring to the user behavior. Namely, it sets the checking time and the open probability from the statistical generator. After this constructor, the user application must be attached to a node, in order to relate physical nodes to the user application. These use cases are presented at Figure 5.3.

![Figure 5.3: Use cases for new user and association with physical machine](image)
5.3.2 Email worm use cases

Worm queries name server for SMTP IP address

Once the email worm is attached to a node, the GTNetS scheduler sequentially activates worm in order to start send infected copies to the potential victims at its list. However, our worm model has guessing capabilities. Therefore, between each successful attack to a victim we model failed attacks in form of a constant ratio. Prior to sending emails, the worm asks the DNS for the IP address of the victim SMTP server. If the list of potential victims is empty the worm does not reschedule itself anymore but does not stop itself. This permits the worm receiving more packets which might include delayed responses and other operations. Once the packet is send, the worm reschedules itself to treat next victim at its list. This use case is presented at Figure 5.4.

Figure 5.4: Worm time out use case: querying the local name server for victim SMTP address

Worm receives the name server response

The worm might receive responses from the name server relating queries it perform before. When the packet is received it is parsed in order to verify if response is not a response error (name server did not find the SMTP server of the queried id). If name server finds the SMTP server the worm sends to it a packet containing an infected copy. This use case is presented at Figure 5.5.

New worm and attach to physical node

When the worm instance is created it only sets the guesser counter to 0. However, when attached to a physical node it builds the social network taking the social network id stored at that node. Afterwards the guessing capabilities of the worm take place. In particular, we add nodes from the social network to the victims list until the least reaches the degree defined by the user. This use case use presented at Figure 5.6.
5.3. USE CASES

Figure 5.5: Worm receives packet use case: receive local name server response and send infected copy

Figure 5.6: Start worm application at host use case

5.3.3 Name server use-cases

Name server receives a worm query

Name servers receive queries in order to resolve names. We provide figure 5.9 to clarify this use case. Normally the name served would response after getting the resolved address from its cache or by querying other servers. However, we model this as explained in our approach. We add retrieve a delay time from the statistical generator. In addition, we update how many queries already performed the client and apply a user-defined latency to the response if defined. Afterwards a new event is scheduled to run at resultant time in order to respond the user.
Name server responds the worm

This use case sends to a certain client the response. From the list of not still responded queries, we retrieve the query packet which lower associated time-stamp. Note that this time-stamp refers to the time where packed should be treated, which value is set at the receive query use-case. After selecting the packet, this is parsed in order to retrieve the victim id. The SMTP server IP address is obtained through the DNS tree class, which contains a list of all nodes at simulation and the SMTP servers they are related to. If no SMTP server is found the query payload reveals an error, otherwise the payload is filled with the SMTP address of the queried victim. Finally, if the client perform more queries than allowed, no response is sent to the client. Otherwise the DNS sends a packet back to the user with the queried information. This use case is presented at Figure 5.7.

![Diagram](image)

Figure 5.7: Name server timeout response use case: send response back

5.3.4 SMTP server use-cases

SMTP server receives a query

SMTP servers might receive packets either for incoming emails either for user queries. Thus, when receiving a packet, this must be parsed in order to apply different functionality. If the packet refers to an incoming email some statistics are stored for later use and the counter of infected emails at inbox is incremented. If the query is from an email user query, the server responds sending a packet with the number of infected nodes at that moment and this number is set to zero because the user might not open those emails again. This use case is presented at Figure 5.8.
Figure 5.8: SMTP server receives packet use case: incoming user query or email

Figure 5.9: Name server receives packet use case: prepare response
Chapter 6

Experiments

6.1 Test cases

In this section we describe the test cases applied in order to ensure proper outcomes.

6.1.1 Functional test cases

We design a small suit of test cases in order to ensure proper function of unitary processes. Namely, we design functional tests for each unitary process in our simulation.

All worm cycle is checked to be performed properly. In particular, worm applications run ensuring that all social network and guessed nodes are queried to the workstation local name server and receive an answer. Namely, we check if they receive an IP address if record exists and an error if it is false guessed. Furthermore, we assure in case of IP address that the IP address correspond to the SMTP server of the potential victim.

We check that name servers correctly access to the information of the DNS tree and send back responses with the proper delay, including normal DNS delay and latency penalties. In addition, it is checked that the DNS tree stops responding queries if defined. Finally, we check that name servers correctly limits the number of incoming queries if a limit is surpassed and enqueues these queries to be processed after the time unitexts.

SMTP servers are ensured to receive and update information about mail and respond to the user in accordance with this information. User model is verified to query the SMTP as modeled and receive answers in function of their mail box. In particular it is checked start executing the worm application if they query the SMTP server and an infected email is there and they click the email.

No test cases were designed for topologies, as if topologies contain errors packets are not routed and therefore lost. Thus, we use GTNetS memory report to detect topology errors. Finally, all functional test cases are grouped to a test case, that assures the correct behavior of the worm life cycle.

6.1.2 Experiment evaluation

We found more difficulties to perform large-scale simulations due to implementation errors rather than functional errors. Namely, it is remarkable the impact of memory leaks into large scale simulations as the scale of it boosts the memory requirements. Memory leaks were apparent as our memory requirements grew up above 3GB for certain types of topologies. Current operative system kernels can not handle a single process with more than 3GB of memory and thus some runs crashed. The authors in [Rile03] state that his simulations with 100000 nodes require around 1GB memory. We used Valgrind to locate memory leaks and current simulations require now no more than 1GB memory, and in some cases, less than 700Mb.
Finally, in order to assure a correct global behavior, we set a global test case consisting of a set
of user machines and a social network where machine $i$ knows machine $i + 1$. At the simulation
machine 0 is infected. Each of these user machines are located in different ASs. Namely, one ASs
has one user machine, one name server and one SMTP server. We check that the epidemic follows the
expected behavior by infecting the end user machines in a row and all component instances behave
as expected.

6.2 Common parameters

In order to reduce variability and obtain more accurate results relating to the experiment topic, we
define a set of parameters for the simulations. In particular:

i. Social network: we use the same social network in [ZoTG04] to reduce variability.

ii. Worm modeling: we use the same worm model in all simulations to reduce variability.

iii. Network components: we different SMTP and NS models for each different experiment.

iv. User behavior: all simulations have the same user model as modeled in [ZoTG04].

v. Physical topologies: the simulations share some topology parameters.

6.2.1 Social network

The social network is formed by a total of 100000 nodes. Social network nodes are randomly assigned
to one physical node in order to avoid having in different runs the same path path length for each par
of social network nodes leading to similar epidemic curves.

The social network was generated with parameters at Table 6.1 with the GLP model. However,
we use the social network at [ZoTG04] for comparability reasons.

<table>
<thead>
<tr>
<th>(a) Social network general parameters</th>
<th>(b) Worm model parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Node Degree</td>
<td>8</td>
</tr>
<tr>
<td>Power law coeff.</td>
<td>-1.7</td>
</tr>
<tr>
<td>Reciprocity</td>
<td>1</td>
</tr>
<tr>
<td>Clustering</td>
<td>-</td>
</tr>
<tr>
<td>Time between packets</td>
<td>0.5s</td>
</tr>
<tr>
<td>Time till worm starts</td>
<td>0.0s</td>
</tr>
<tr>
<td>Positive guessing ratio</td>
<td>50%</td>
</tr>
<tr>
<td>False guessing ratio</td>
<td>9</td>
</tr>
<tr>
<td>#initial infected nodes</td>
<td>2</td>
</tr>
</tbody>
</table>

6.2.2 Worm model parameters

Each simulation starts with 2 random initial infected nodes, whose node degree is in a range degree
between the 50% and the 95% most connected nodes. This constraint has as objective avoiding the
selection of high-degree or low-degree nodes as initial nodes because as stated per [ZoTG04], initial
degree can drastically change the mean curve.

We assume the worm to send emails to the addresses as it finds them while harvesting. Therefore,
it does not have a wait time since it user machine is infected, but has a time delay between packets
representing the time a worm needs to find another potential victim address.

We assume the worm to have guessing capabilities. In particular the worm is able to successfully
guess a set of victims sized to the half of its social network as stated in [Xion04]. However, a worm
can success or not in this attempt. We rate a success ratio of one every nine tries. We set this parameter as a result of a small experiment.

The experiment counts the number of DNS queries a worm performs in a typical corporate network within a minute. The authors in [WBMW04] report the number of emails that an email worm sends per minute. We can obtain the number of DNS queries that the worm does not use with this two parameters and we model this number as the number of false guessing ratio.

We list the parameters of the worm model at Table 6.1.

### 6.2.3 Network components and user behavior

We simulate both local name servers and SMTP servers functionality. DNS response time is modeled as a normal distribution extracted from the Fraunhofer Fokus dataset. Both SMTP and DNS servers have no overload. The SMTP does not implement any mechanism to detect infected emails. Therefore all incoming email messages are accepted. Refer to Table 6.6 for all parameters.

The experiments user model is defined as in [ZoTG04]. Refer to Table 6.2(b) for more details.

<table>
<thead>
<tr>
<th>Table 6.2: Email user, SMTP and name server common parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) SMTP and DNS server model parameters</td>
</tr>
<tr>
<td>NS Response time mean</td>
</tr>
<tr>
<td>(b) Email user model parameters</td>
</tr>
<tr>
<td>Checking time mean</td>
</tr>
<tr>
<td>Opening prob. mean</td>
</tr>
</tbody>
</table>

### 6.2.4 Physical topologies

#### Physical topologies links

We consider using default stub networks links bandwidth and delays for the inside AS topology as the default link parameters for the GTNetS simulation tool [Rile03]. All ASs, independently of which category they belong to, have the same links. Links between ASs have also same bandwidth and delay. This approach is not realistic because real-world link capacities are diversified. However, this is enough to validate the effects of underlying physical topologies into the worm spreading. Parameters are listed at table 6.3(a)

#### AS topology

In order to model the Internet backbone we previously defined many topologies. However, when considering realistic topologies we only consider the connection of ASs in a power law. We generate such topologies with the power law generator GLP and parameters as reported in [BuTo02], which are listed at Table 6.3(b).

### 6.3 Physical topologies and spreading dynamics

#### 6.3.1 Motivation

**Topic**

One of the objectives of the thesis is to verify if the underlying physical topology directly plays a role on the email worm spreading dynamics.
6.3. PHYSICAL TOPOLOGIES AND SPREADING DYNAMICS

Table 6.3: Bandwidth, delays and power-law backbone parameters

<table>
<thead>
<tr>
<th>(a) Link delays and bandwidth</th>
<th>(b) Power-law Internet AS generator parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-AS Link Delay</td>
<td>Coefficient</td>
</tr>
<tr>
<td>Intra-AS Link Rate</td>
<td>-1.155</td>
</tr>
<tr>
<td>Inter-AS Link Delay</td>
<td>Avg.degree</td>
</tr>
<tr>
<td>Inter-AS Link Rate</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>m (avg. initial deg)</td>
</tr>
<tr>
<td></td>
<td>1.13</td>
</tr>
</tbody>
</table>

**Justification**

Worms rely on certain events and components with latencies: infected host network, the name server, the Internet backbone, the SMTP server network, the SMTP server, the target victim network and finally the target victim checking time latency. All of them apart the user latency and the servers latencies are directly related to the delay of the communication links.

The authors in [ZoTG04] argue that the user email checking time latency magnitude is much bigger than the underlying physical network latency magnitude and therefore it can be neglected. However, latencies are not stable through time, and abnormal transmission latency can be observed at congested networks, networks with old infrastructures, overloaded servers, etc.

In normal conditions, a modern network without congestion and all nodes up, each of these latencies are in the magnitude of milliseconds. Thus, we assume as [ZoTG04] do, that in normal conditions the network topology might not affect the spreading of a worm dynamic because the user checking time is much bigger in magnitude than the other latencies. However, we proceed to simulate a worm incident over different network topologies, in order to validate this assumption and observe the effect of the topology configuration in normal conditions at these time magnitudes.

**Approach**

We use the Georgia Tech Simulator, GTNeTS, and run it with different underlying physical topologies to obtain different metrics relating the spreading curve of each different configuration. DNS and SMTP are modeled as proposed in the approach chapter. We model the user behavior and use the same social network used at [ZoTG04].

Our topology models are formed per AS networks, or stub networks, which provide connectivity to all end-hosts and have a gateway to the backbone network. Therefore, we compare the topologies at two levels. Firstly, we compare the impact of the stub network topology with the same backbone topology, and then we compare the impact of the backbone topology with the same stub network topology.

**6.3.2 Scenario configuration**

**Simulation topologies**

We propose simulating five different topologies from those presented at the approach chapter. We simulate a total of 200 AS with 504 hosts each one, reaching a global number of 100800 end-hosts, 200 DNS servers and 200 SMTP servers. Each AS hosts a DNS and SMTP server. End-hosts and worms are modeled to query the topologically nearest SMTP and DNS server. Topologies are listed at Figure 6.3.2. We compare topologies at two levels:

i. We compare how the topology inside an AS affect the propagation. Namely we compare three stub network topologies with the same backbone topology. We model the backbone topology
as a power law with parameters found at [BuTo02], and ASs as a set of stars connected as full graph (FullG), ring graph (RingG) or as a small campus network (Campus)[FPPW03].

ii. We compare how backbone topologies affect the worm propagation by comparing three topologies at the backbone level. We connect all ASs in full graph, ring graph or as a power law network (PowerLaw) [FaFP99]. The AS stub network topology is a set of stars connected as a full graph.

Table 6.4: Topology configurations and delay since worm finds a potential victim address and the SMTP server receives an infected email

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Config. 1</th>
<th>Config. 2</th>
<th>Config. 3</th>
<th>Config. 4</th>
<th>Config. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stub networks</td>
<td>FullG</td>
<td>FullG</td>
<td>FullG</td>
<td>Campus</td>
<td>RingG</td>
</tr>
<tr>
<td>Intraconnection</td>
<td>FullG</td>
<td>RingG</td>
<td>PowerLaw</td>
<td>PowerLaw</td>
<td>PowerLaw</td>
</tr>
<tr>
<td>Delay (ms)</td>
<td>2,19</td>
<td>26,57</td>
<td>3,68</td>
<td>4,45</td>
<td>4,43</td>
</tr>
</tbody>
</table>

Topologies diameter

Figure 6.3.2 presents the different topology configurations and the average delay to send an infected copy. We obtain the average delay by counting the number of hops from every host in the network to reach its local name server and going back to the host, plus adding the number of hosts to reach the infected host network gateway if target is outside domain, plus the hops needed to reach the SMTP server AS, plus the hops needed to reach the SMTP server from the AS gateway. Figure 6.5 provides path length in time or hops for each of the previous steps.

We observe that differences are in the magnitude of milliseconds, what in our model validates the assumption that in normal conditions the latency penalty resultant of the physical topology is very small compared to user latencies.

Table 6.5: Path measurements: hops and delays

<table>
<thead>
<tr>
<th>Stub networks</th>
<th>FullG</th>
<th>Campus</th>
<th>RingG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average hops</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host-Gateway</td>
<td>4</td>
<td>5,91</td>
<td>6,5</td>
</tr>
<tr>
<td>Host-NS</td>
<td>5</td>
<td>6,91</td>
<td>7,5</td>
</tr>
<tr>
<td>Host-SMTP</td>
<td>5</td>
<td>7,91</td>
<td>7,5</td>
</tr>
<tr>
<td>Gateway-SMTP</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Average delay (ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host-Gateway</td>
<td>0,4</td>
<td>0,59</td>
<td>0,65</td>
</tr>
<tr>
<td>Host-NS</td>
<td>0,5</td>
<td>0,69</td>
<td>0,75</td>
</tr>
<tr>
<td>Host-SMTP</td>
<td>0,5</td>
<td>0,79</td>
<td>0,75</td>
</tr>
<tr>
<td>Gateway-SMTP</td>
<td>0,3</td>
<td>0,5</td>
<td>0,3</td>
</tr>
<tr>
<td>Intraconnection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. Bbone Hops</td>
<td>1</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>Avg. Bbone Delay (ms)</td>
<td>0,5</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>
6.3. PHYSICAL TOPOLOGIES AND SPREADING DYNAMICS

6.3.3 Results

Inter-AS topology results

Results presented are obtained after running 100 times each different topology. We present measures for the number of infected node with different AS topologies at Figures 6.1. We observe that the mean curve reflects how AS topologies with larger diameter have slower propagation.

As our number of samples is relatively low, we verify the variance of the samples. The figure 6.2 reports the median epidemic curve for the compared topologies. We obtain a variance between 16000 and 18000 nodes in each case, which led us to conclude that too much variability was introduced. This variability is caused by the initial phase of the worm incident as pointed by [ZoTG04]. The variance magnitude is much larger than the differences between mean values and therefore, we assume that AS topologies might not have an effect into the worm spreading if their latency magnitude is in range of milliseconds.

![Figure 6.1: Mean epidemic curves with full connected stub network topology and different backbone topologies](image)

Intra-AS topology results

Results were obtained after running 50 times each different configuration. Again we present metrics for the number of infected nodes with different backbone topologies at Figure 6.3. The figure 6.2 reports the median epidemic curve for the benchmarked topologies. Results are not conclusive because of the variance of the samples, being their range between 16000 and 18000 nodes again. Therefore, we assume that the variability of the worm incident has a more important role than the topology itself in term of network latencies.

Other metrics results

At Figure 6.5 we present other metrics for the power law backbone topology with small campus networks as stub networks. We consider this the most realistic topology of the above presented. We provide the metrics for the number of infected nodes, the number of active worms and the number of susceptible nodes within a time unit.
Figure 6.2: Median epidemic curves with full connected stub network topology and different backbone topologies

Figure 6.3: Mean epidemic curves with different stub network topologies and power law backbone topology
6.3. PHYSICAL TOPOLOGIES AND SPREADING DYNAMICS

Figure 6.4: Median epidemic curves with different stub network topologies and power law backbone topology

Figure 6.5: Other metrics for power law backbone with campus networks as stub networks
6.4 Email worm containment by delaying name server responses

6.4.1 Motivation

Topic

This experiment intends to measure the effect of delaying name server responses into a worm incident spreading. Namely, our assumption is that delaying a name server response might add a time weight to each social network link, and thus, slow down the worm spreading into the social network.

Justification

Results in the previous experiment show that the underlying topologies do not have a relevant impact into the worm spreading when they are in a magnitude of milliseconds. It is not possible to increase the delay at the network topology for connectivity reasons, but it is possible to do this at the application components involved in the worm spreading. Thus, this experiment intends to check the effects of longer delays into the worm spreading.

Approach

We propose simulating a worm incident where name server responses are delayed. In particular, time penalizations are in range from seconds, being the extreme case as long as the default timeout of the Unix command dig, 5 seconds. We argue that longer latencies are not applicable because the user or worm might re-query the name server after the timeout of the stub resolver. Refer to Table 6.6(b) for a detailed explanation.

The simulation network is build from 200 ASs, being each one a campus network, connected as a power law. The basic NS response time is calculated from a normal distribution and we add the latency penalization for each experiment.

<table>
<thead>
<tr>
<th></th>
<th>(a) NS response time penalization</th>
<th>(b) Normal NS response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>5s</td>
<td>Response time mean</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>2.5s</td>
<td>Response time std. deviation</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>1.75s</td>
<td>0.088s</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>0.875s</td>
<td>0.401s</td>
</tr>
</tbody>
</table>

6.4.2 Results

Metrics

Figure 6.6 shows the number of infected nodes per time. It clearly shows that latencies at the name server do not have an impact into the worm dynamics. Furthermore, the figure shows the higher number of infected nodes for the highest latency penalization. We argue that the variability of worms incidents plays a stronger role into the spreading dynamics that a penalty delay in such magnitude.

Figure 6.7 shows the average number of infected nodes after ten hours since the epidemic started when applying different time delays. The figure suggests that the effect of link delays at such magnitude is not effective and shows that these magnitudes are irrelevant.

However, when there is query time-out the worm might retry to query the local name server. We argue that users can not re-query the name server so fast like a worm that automatically retries a query.
This can be seen as a sign of an abnormal user behavior. In other words, by delaying the response time more than the timeout an infected host can be detected more easily. However, this approach is not useful in terms of email worm mitigation.

![Graph showing epidemic mean curves of infected nodes per time](image1)

Figure 6.6: Epidemic mean curves of infected nodes per time

![Graph showing effectiveness of delaying NS responses after ten hours since outbreak](image2)

Figure 6.7: Effectiveness of delaying NS responses after ten hours since outbreak

**Other results**

We assume that delaying responses affect the worm spreading when they are in a scale similar to the checking time scale. Namely, we believe that the probability that a user gets infected within a unit time is not a single-factor parameter. Therefore, we also experiment with higher delays, in order to verify this assumption.
We find in Figure 6.8 that our assumption holds and that the worm propagation time affects the epidemic spreading. We argue that the checking time and the propagation time are independent parameters, and therefore the probability that a user gets infected is somehow related to the proportion between these parameters as we find in 6.9.

Figure 6.8: Epidemic mean curves of infected nodes per time (II)

Figure 6.9: Effectiveness of delaying NS responses after ten hours since outbreak (III)
6.5 Email worm containment by rate limiting name server responses

6.5.1 Motivation

Topic

This experiment intends to measure the effect of rate-limiting name server responses into a worm incident spreading. Namely, our assumption is that limiting the number of queries a name server responds to an infected user within a unit time slows down the worm epidemic.

Justification

Results in the previous experiment showed that delaying all responses with the same penalty does not affect the spreading of a worm virus when this penalty is in the range of seconds. As it is not realistic to delay name server responses more than few seconds, we consider the previous approach not applicable.

We assume that the previous approach is not effective because responses are likely to be treated sequentially. We consider that the previous approach delays the worm start the same amount of time as the delay. Therefore, there is no effect at later states of the epidemic. We assume that if the name server rate-limits the amount of queries within a unit time the name server responses are delayed in a non-constant way, leading to stronger penalizations for those machines performing more queries. Namely, if a worm surpasses the limit, queries are enqueued for further process leading to delays in the order of minutes if the amount of queries between divided by rate-limit is a large number.

This approach is not restrictive as it avoids the penalization of users doing an acceptable number of queries. Namely, a worm might modify the traffic patterns of an user considerably as reported per authors in [WBMW04]. Therefore, we assume this approach to have a good trade-off between reliability and mitigation effectiveness.

Approach

We here describe how the name server rate-limits queries. The name server counts the number of queries within a unit time. When a query arrives it is processed. However, if the user surpasses the query limit the name server does not respond until the next unit time. When the unit time changes, the name server starts processing the enqueued queries if they are apparent, but stops processing when the rate-limit is surpassed.

We propose simulating a worm incident where name server responses are rate-limited within a unit time. The maximal number of queries a worm performs in a small campus stub network within a minute is around 128. We consider this number high for a normal user. Therefore, we simulate an epidemic with other three rate-limit parameters, which are listed at Table 6.7. The simulation network is built from 200 ASs, being each one a campus network, connected as a power law.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>128 queries/minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 2</td>
<td>64 queries/minute</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>32 queries/minute</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>16 queries/minute</td>
</tr>
</tbody>
</table>
6.5.2 Results

Metrics

Figure 6.10 shows the epidemic curves when applying different rate-limit parameters. The figure shows clearly that the rate-limiting strategy is effective. Furthermore, it shows that limiting the number of queries a user can perform within a minute to sixteen has a deep impact into the spreading. The human behavior is likely to not surpass this amount of queries within a minute. We therefore argue this approach to be applicable for email worm mitigation. In addition, although we here apply static rate-limits, we argue that this approach can be dynamic depending on user previous statistics.

Figure 6.11 shows the effectiveness of rate limiting. In particular it shows the number of infected nodes after ten hours as we decrease the rate-limit. The figure shows a non-linear behavior of the effectiveness, suggesting that the power-law topology of the social network plays a role into this metric.

![Epidemic mean curves of infected nodes per time](image)

Figure 6.10: Epidemic mean curves of infected nodes per time

Other results

In this section we present other metrics. Figure 6.12 shows the average degree of the infected nodes. The average degree is relatively higher at a later stage in rate-limited epidemics. This suggests by rate-limiting the number of queries the boosting effect of the high-degree nodes is controlled, leading to a slow down of the epidemic.

Figure 6.13 presents the number of active worms per time and Figure 6.14 the number of susceptible nodes per time. Figures show how applying a rate-limiting strategy slow down the epidemic as the number of active worms and the number of susceptible nodes are lower as the rate-limit becomes more restrictive.
Figure 6.11: Effectiveness of rate-limiting NS responses after ten hours since outbreak

Figure 6.12: Average degree of active worms per time
Figure 6.13: Average number of active worms per time

Figure 6.14: Epidemic mean curves of infected nodes per time
6.6 Email worm containment by blocking name server responses

6.6.1 Motivation

Topic

This experiment has as goal proving the effectiveness of blocking DNS queries in order to slow down a worm incident. Namely, our assumption for this experiment is that blocking DNS queries increases the network diameter because social network links are removed.

Justification

In previous experiments we show that weight in terms of time at social network links might impact on the the spreading of the topological worm. While this approach might be useful in some contexts, the worm incident is still able to reach all nodes at the social network.

However, if social network links are disabled, we assume that the topology diameter of the social network decreases and furthermore, it is possible that the social network fragments into disconnect components. Therefore, we propose to modify the structure of the social network topology by removing social network links through the name server. Namely, a name server can disable social network links because it provides this information to the worm.

Other approaches propose to remove social network nodes, and in consequence all the links associated, through selective immunization [ZoTG04]. We argue that for being effective, large parts of the social network topology must be known in order to determine which are nodes to be immunized. However, this is difficult to perform because socials networks have strict privacy policies. In addition, we argue that immunization requires developing a cure for the epidemic before the epidemic reaches the selected hosts. Worm incidents reach epidemic levels in question of hours. We point the difficulty for humans to develop a signature for the worm incident in such time windows.

A DNS query blocking strategy allows reducing the degree of infected nodes and slow down the epidemic. Namely, the number of out-going links at high-degree nodes is limited. In contrast to selective immunization, this approach might be applicable to all nodes at the social network, depending only of the pattern detection algorithm effectiveness.

Approach

We simulate a worm incident where the local name server blocks DNS queries for a user $z$ after $x$ queries. Namely, the detection mechanism is capable of detecting a worm after $x$ queries, and therefore we block it. We mine our social network for an ordered list of the degree. We consider limiting the maximal degree in proportion to the maximal degree of the social network Results are listed at Figure 6.8. We set the number of maximum queries an infected host can perform as a $n + g(n - 1)$ where $n$ is the degree of the the node with higher degree (including positive guesses) and $g$ is the number of false guessing queries.

6.6.2 Results

Figure 6.15 measures the impact of the degree limitation in the spreading curve of the worm incident. Namely, figure shows how blocking the number of queries of a node has a very strong effect on the spreading dynamics. It also suggests that the relative effectiveness ratio of the mechanism reduces as the number of limited nodes increases. In other words, node degree limiting is more effective in power law topologies that in uniform degree topologies.

To support this assumption we provide Figure 6.16. It shows the number of infected nodes after ten hours at axis $y$ and the query rate applied at axis $x$. The figure shows that for our social network
the effectiveness, or function decrement, of the mechanism decreases in proportion as we augment the percentage of limited nodes.

However, figure 6.17 depicts how the average degree at the limited degree curve is higher than the not limited degree curve at later stages. This happens because high degree nodes might be lately infected. Therefore, there might be less active worms, but with higher degrees at later times. This is in concordance with the power law nature: infections tends to affect firstly high degree nodes and thus the average degree is higher at earlier stages.

Figure 6.18 shows the number of emails sent at a time $t$ at the system. Namely, the figure shows the traffic generated by the worm in terms of email. We consider two cases: not limited epidemics and epidemics with queries limited to 301. It clearly shows that the blocking strategy avoids polluting the network.

Figure 6.19 shows that the number of susceptible nodes does not increase so fast when limiting the node degree. We explain this because restricting the degree reduces the number of effective infection links, which create new susceptible nodes.

Figure 6.20 depicts the number of active worms at a time $t$. The figure shows that the number of active worms is moreover stable when not limiting the degree. It also shows that when limiting the degree the number of active worms is relatively small and the number of active worms has not still grow up exponentially. This means that a blocking strategy has strong effects into the spreading dynamics of an email worm.
Figure 6.16: Number of infected nodes after 10 hours of epidemics with discard ratio

Figure 6.17: Average degree of active worms per time
Figure 6.18: Generated traffic per time

Figure 6.19: Number of susceptible nodes per time
Figure 6.20: Number of active worms per time
Chapter 7

Final plan and cost analysis

7.1 Final plan

The Gantt diagram relating to the thesis final work-road is presented at Figure 7.1. It differs substantially from original plan, because the research phase took more time as initially planned.

Plan deviations due to complexity

We initially identified four main fields of study: worm epidemics, network topologies, Domain Name System and simulations. The amount of apparent work and the complexity of some of these fields derived into a necessity for a time extension which was not initially planned in order to assure matching the state of the art on such fields.

Worm epidemics is a topic of relevant difficulty. Namely, epidemic theory is a topic where mathematics are strongly involved. As we pretend to explain epidemics through simulation and propose mitigation mechanisms, knowledge of the involved mathematics was absolutely necessary and therefore more time was needed than the initially planned.

Plan deviations due to implementation

Implementation errors and the learning curve of GTNetS are two relevant causes for time deviations. Namely, memory leaks have a very strong impact into the memory requirements of large-scale simulations. We initially did not expect such behavior and specially tools were needed to identify and remove such errors. Therefore, the debugging phase took more time than initially expected.

In addition, the learning curve of GTNetS required more time than expected. GTNetS is a simulator environment built to run large-scale simulations at fine level, which means that it requires a better understanding of networking protocols and components than other simulators.

7.2 Project costs

7.2.1 Hardware and software

Hardware

Table 7.1 lists the hardware and software used in this topic. We cope all software requirements with software under GPL license. Therefore our software costs are null. However, in order to achieve fast results we needed a dedicated computed machine. Due to the number of simulations initially planned (around two thousand), Fraunhofer Fokus acquired a machine with 8 processors to allow parallel and distributed simulations. Therefore we assume this machine to be amortized in a five year period with other computing projects, as well as the machine used as a desktop equipment.
Table 7.1: Hardware and software costs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Cost</th>
<th>Amortization period</th>
<th>Project cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Machine</td>
<td>Dual Quad-Core Intel Low Volt Xeon 1.5310 processors at 1.6GHz 4GB RAM Rack</td>
<td>2.596</td>
<td>5 years</td>
<td>207,68 €</td>
</tr>
<tr>
<td>Workstation</td>
<td>Pentium 4 2.0GHz 512MB RAM Desktop</td>
<td>939</td>
<td>3 years</td>
<td>104,33 €</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td><strong>3.535 €</strong></td>
<td></td>
<td><strong>312,01 €</strong></td>
</tr>
</tbody>
</table>

**Software**

It is policy of the institute to use Open Source software when applicable. Therefore in this thesis we use only open source software, mostly distributed under the GPL license. We list the software used at table 7.2.

Table 7.2: Software

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 6.10 Server</td>
<td>GNU/Linux OS for Magpie</td>
</tr>
<tr>
<td>Xubuntu 6.10</td>
<td>GNU/Linux OS for Valkris</td>
</tr>
<tr>
<td>GCC</td>
<td>Gnu Compiler Collection</td>
</tr>
<tr>
<td>GTNetS</td>
<td>Georgia Tech Network Simulator</td>
</tr>
<tr>
<td>GSL</td>
<td>Gnu Scientific Library</td>
</tr>
<tr>
<td>GnuPlot</td>
<td>Plotting tool</td>
</tr>
<tr>
<td>GLP</td>
<td>Degree-based graph generator</td>
</tr>
<tr>
<td>NetMine</td>
<td>Graph mining tool</td>
</tr>
<tr>
<td>Perl</td>
<td>Scripting language</td>
</tr>
<tr>
<td>Valgrind</td>
<td>Debugging tool</td>
</tr>
<tr>
<td>GDB</td>
<td>Gnu Debugger</td>
</tr>
<tr>
<td>Kile</td>
<td>Latex editor</td>
</tr>
<tr>
<td>Dia</td>
<td>Diagram tool</td>
</tr>
<tr>
<td>GanttProject</td>
<td>Gantt diagrams tool</td>
</tr>
</tbody>
</table>

**7.2.2 Human costs**

Table 7.3 list the human resources costs for this thesis. We define three profiles and different salaries for each one.

i. **Project manager**: person in charge of requisites accomplishments, control points and delivery dates. Nikolaos Chatzis from the Fraunhofer Fokus research Institute took the role of project manager for this thesis.

ii. **Analyst**: performs the requirement analysis, functional specification and system design.
iii. *Programmer*: implementation of the system. We count the research phase as an analyst task. However, it is not strictly a task of the analyst, but in our case is heavily related. Table 7.3 does not reflect other costs as thesis document writing, project preparation, slides elaboration between others.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Hours</th>
<th>Cost/hour</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>82</td>
<td>58,30 €</td>
<td>4,780,60 €</td>
</tr>
<tr>
<td>Analyst</td>
<td>415</td>
<td>44 €</td>
<td>18260 €</td>
</tr>
<tr>
<td>Programmer</td>
<td>386</td>
<td>20,50 €</td>
<td>7913 €</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>883</strong></td>
<td></td>
<td><strong>30,953,6 €</strong></td>
</tr>
</tbody>
</table>

### 7.2.3 Total costs

Table 7.4 lists total costs for this project in three categories: hardware, software and human costs.

<table>
<thead>
<tr>
<th>Resource</th>
<th>SUBTOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>312 €</td>
</tr>
<tr>
<td>Software</td>
<td>Not applica-ble</td>
</tr>
<tr>
<td>Human</td>
<td>30953,6 €</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>31,265 €</strong></td>
</tr>
</tbody>
</table>
Working environment setup
Related work
  DNS functionality and studies
  Social networks
  Physical topologies
  Worm
Fokus dataset
  Authoritative name servers
  Response time model
Simulation tools survey
Requirement analysis
  GTNetS constraints
  Use cases
  Class diagram
Implementation
  GTNetS learning curve
  Graph generation
  Topologies implementation
  Auxiliary components
  Application layer implementation
Debugging
  Optimizations
  Global test cases
  Memory leaks
Execution @ Magpie
Results parsing scripts
Diploma thesis document

Figure 7.1: Final plan Gantt diagram
Chapter 8

Conclusions and future work

8.1 Conclusions

Name server model

We provide results and a model derived from a medium-size research institute. We model the response time as a normal distribution. We report advantages for such approach, such avoiding to model an hierarchical overlay network and leveraging the memory requirements at the simulation.

In this thesis we present the results of mining a three day DNS dataset from a medium-sized research institute and contribute to the research community with a name server response model for such organizations.

Physical networks and email worms

The underlying physical topology latency can be neglected for email worms incidents. Email worms are characterized by the user checking time, which latency magnitude is much bigger than physical network latencies. In addition, the activation mechanism is an independent process which does not relate to the arrival of an infected email.

At this thesis we validate the previous statement through experiment number one; and verify that network latencies do not play a role into the an email worm incident spreading.

Social networks edges weight

A social link latency depends on the underlying physical topology. If latencies at the social network links are in the magnitude of activation latencies the worm epidemic might be affected and worm propagation might slow down. It is possible to modify latencies at the social network by controlling infrastructure components of the underlying network. However, such effects might be only effective when social link latencies are only in the same magnitude as the activation mechanism.

At this thesis we validate this assumption through experiment number two; and propose an email worm mitigation mechanism based on delaying responses at the name servers. However, we verify that a reasonable delay, in the order of seconds, is not effective. However, we prove at experiment two that the email worm propagation and email worm activation are two independent processes that if they have similar magnitudes they strongly affect the worm spreading. Therefore, we suggest considering social networks as weighted graphs in cases were the activation latency similar to the social network link latency.

Slowing down the boosting effect of power-law topologies

The spreading of diseases in power law networks has been widely studied in related literature, and report that power-law networks boost the spreading of diseases due to intrinsic characteristics. While
some approaches to mitigate diseases in such networks propose deleting the most connected nodes in order to cancel this effect, we assume that rate-limiting the number of actions an infected node can perform within a unit time minimizes the boosting effect of power-law topologies, leading to a slower epidemic.

At this thesis we validate this assumption through experiment number three; and propose an email worm mitigation mechanism based on rate-limiting the number of DNS queries a machine can do within a unit time. We verify that email worms considerably slow down their propagation.

Disabling the boosting effect of power-law topologies

As social networks are build upon physical networks, we assume that it is possible to modify the structure of the social network through physical infrastructure components. Previous work states an epidemic might be slowed down when modifying the percolation threshold. We propose using infrastructure components as the name servers to modify the percolation threshold, in a similar way as selective immunization does.

As name servers do not have a global knowledge of all the social network, we propose limiting the outgoing number of links of a social network node. Namely, we assume that blocking the DNS queries from emails worms partially disables the amplifying factor of scale-free networks by limiting the degree of high-degree nodes.

At this thesis we validate this assumption through experiment number four; we approach this issue by limiting the number of DNS responses after a name server detects an email worm. We verify effectiveness of the mitigation mechanism based on removing social network links by limiting social networks high-degree nodes.

Pollution control for email worms

Current mitigation mechanisms do not avoid the network being polluted with email worms traffic. We assume that mitigation mechanisms at components involved into the worm potential victims discovery process leverages the outgoing traffic of email worms, leading to better performance of the network. Furthermore, we argue that mitigation mechanisms between infected machines and potential victim machines provide an additional layer of security, which is nowadays not apparent for email worms.

At this thesis we validate this assumption through different experiments, and verify that the amount of infective traffic can be reduced through mitigation mechanisms at the name server.

8.2 Future work

Physical network topologies

We perform experiments over an degree-based AS topology and place the same stub network to every AS node. This assumption does not reflect the heterogeneity of ASs, where sizes and topologies vary. We suggest that email worm spreading might be affected by the AS taxonomy when mitigation mechanism are applied to the name server. Namely, we assume big ASs to notice earlier and with more accuracy worm incidents than small ASs. Further work is required to validate this assumption.

In addition, we assume that every AS provides a NS and a SMTP server. However, this might not hold in reality, as the DNS name tree does not strictly hold a one-to-one relationship with the AS, but a none-or-many-to-one-or-none. Thus, we propose as future work to model more accurately DNS domains within an AS in order to obtain more accurate results.

Finally, we assume that users query the SMTP server located at their AS. However, this might not happen in the real world as users are not static and check their email remotely. Thus, it is necessary to accurate SMTP designation for every user.
Email user models

We model email users with simple statistical distributions. However, this does not hold in the reality. Network telescopes reflect how email worms follow the same diurnal and nocturnal patterns as users. Namely, such telescopes have observed a significant decrease of the infection rate at a given region during night hours.

At this thesis we propose mechanisms capable of delaying epidemics until night hours, which provides more time in order to develop a signature-based control mechanism. Therefore, we encourage future work on user modeling in order to provide more accurate models, which might be used to study the spreading of epidemics in different time-regions, and analyze the propagation of email worms in social networks with more variable activation mechanisms.

Email networks and email users

The social network used in this thesis has been built with parameters obtained from an email provider social network. In the real world users belong to more than one social network. Namely, we argue that users have several email accounts, relating to their job and private life. Therefore, the social network of an user is composed by more than one email network. The email network here used is probably an underestimation, resulting in smaller degrees. Therefore, we encourage further work on this topic to provide more realistic email networks.

We also suggest that other users might be involved on the same host. Namely, members of the same family or company which accessed previously to the host machine and whose email contacts might be harvested by a single worm instance. In consequence, we expose the possibility that our results underestimate the spreading of the worm incident and recommend further work on the topic in order to verify or correct mitigation mechanisms.

Email networks and email worms

We consider accounts to be uniformly distributed through DNS domains. However this might not hold in reality as some domains such large email providers might have millions of users, while others few. If worms perform DNS caching or IP addresses distribution to newly infected nodes, email worm mitigation mechanisms relating DNS might be overestimated.

We suggest that the email social network from the DNS point of view is a social network were nodes are domains resulting on a smaller social network. We encourage further work on the clustering properties of email users into domain names, in order to evaluate accurately the effectiveness of DNS based mitigation mechanisms.