UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zrinka Puljiz

DISTRIBUTED REAL-TIME SIMULATION OF HIGH SPEED IP NETWORKS

DISTRIBUIRANA SIMULACIJA BRZIH IP-MREŽA U STVARNOM VREMENU

MASTER THESIS
MAGISTARSKI RAD

Zagreb, 2007.

Magistarski rad izrađen je na Zavodu za telekomunikacije
Fakulteta elektrotehnike i računarstva

Mentor: prof. Miljenko Mikuc

Magistarski rad ima: 117 stranica

Magistarski rad broj:

Povjerenstvo za ocjenu:

1. Prof.dr.sc. Mladen Tkalić, predsjednik

2. Prof.dr.sc. Miljeko Mikuc, mentor

3. Doc.dr.sc. Darko Huljenić, Ericsson Nikola Tesla Zagreb

Povjerenstvo za obranu:

1. Prof.dr.sc. Mladen Tkalić, predsjednik

2. Prof.dr.sc. Miljeko Mikuc, mentor

3. Doc.dr.sc. Darko Huljenić, Ericsson Nikola Tesla Zagreb

4. Prof.dr.sc. Branko Mikac, zamjenik

Datum obrane: 26.listopada 2007. godine

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past few years interest in network simulators has rapidly grown. The main reason for this is the overwhelming need for test environments in research and development of network protocols. Using real network data offers the most reliable results in that area, but it comes with certain drawbacks, such as the need for economical resources which are commonly not available. Changing the topology or any other parameter of a real network environment reveals its inherent inflexibility. This can be, and often is, reduced by using network simulations.

Network simulators that operate in real time and provide emulation interfaces can be very demanding in terms of CPU cycles or memory usage. But at the same time they give more realistic results and a chance to mix synthetic traffic with real one [53]. In order to reduce this demand the approach used distributes network simulation over an existing topology and therefore preserves simulation benefits. These include reducing the costs of real test environments, providing centralized control of the simulation and using all the available resources to make the simulation more realistic.

The focus of this thesis is on distributing network simulation based on IMUNES. IMUNES is a network emulator that operates in real time and offers a very good scalability (10s to 100s of nodes on one commodity PC).

There are three three main areas of contribution:

- Design of a new algorithm for Internet-like topology generation.
- Usage and performance comparison of different graph partitioning algorithms on the IMUNES topology.
- Design, implementation and evaluation of a decentralized distributed network

emulator.

In the case of large network simulations, the process of defining a network either by using GUI or writing a configuration file can be very time demanding and error prone. As a result, the need for building a network topology generator that would speed up the process was identified. The network generator should provide support for building Internet-like topologies with a given number of nodes.

Division of the topology into disjoint pieces that can be simulated on different machines takes place once a large network topology for testing is created. Graph partitioning algorithms can be useful when dealing with this problem. Further on, a performance comparison between two graph partitioning algorithms is given.

The most significant contribution of this thesis is building a distributed network environment for network emulation. Four key management components are identified and implemented as a prototype of the system. The system is designed to be decentralized, transparent and secure.

The road map for this thesis is as follows. In Chapter 2 there is an overview of IMUNES. The topology generation is presented in Chapter 3. In Chapter 4 the focus is on the graph partitioning algorithms and their usage in network simulation. Chapters 5, 6 and 7 cover the design implementation and evaluation of distributed network simulation environment. Chapter 8 is the conclusion.

Chapter 2

IMUNES

IMUNES is a fast network emulator developed at the University of Zagreb, Faculty of Electrical Engineering and Computing, at the Department of Telecommunications [52]. IMUNES operates in real-time and does the network simulation on packet level using real TCP/IP stack. In this chapter description of IMUNES is given because our distributed emulator is built on top of it.

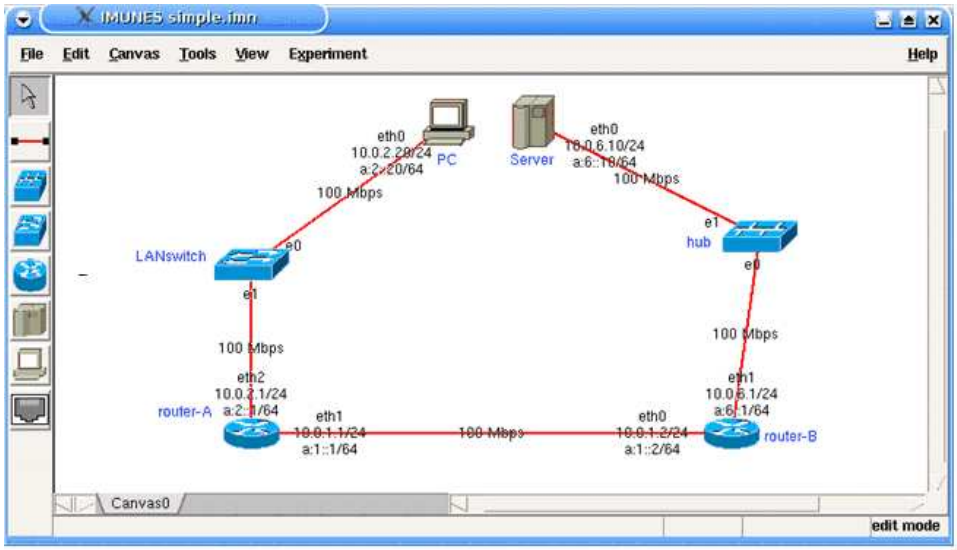
As observed from Figure 2.1, IMUNES has a GUI where a network can easily be defined. IMUNES distinguishes between GUI level objects and kernel level objects. GUI level objects are used for topology specification, and kernel level objects are used for emulation. Kernel level objects are created only upon starting the simulation, and destroyed after the simulation is stopped.

In the following few sections there are IMUNES topology models, properties and the implementation. Except for presenting the background, this chapter can be viewed as a motivation chapter describing IMUNES possibilities and limitations that are overcome in the distributed version.

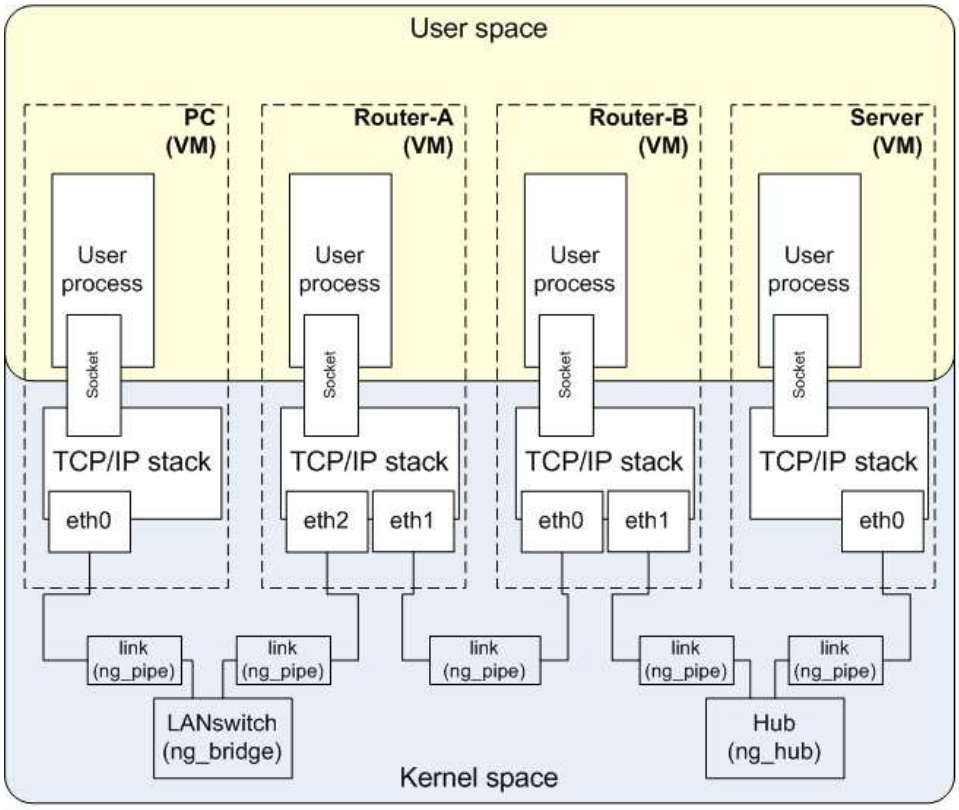
2.1 Network representation in IMUNES

IMUNES uses GUI for topology specification. The design of GUI is presented in Figure 2.2. The topology in IMUNES consists of two basic building units: nodes and links. Nodes can exist independently whereas links always connect two different nodes.

Nodes are entities where packets are created or through which the packets are processed. Depending on the operation executed on the node, nodes are classified into two different groups: link layer nodes and network layer nodes.



a) Topology as specified in IMUNES



b) Topology as deployed in kernel

Figure 2.1 IMUNES objects

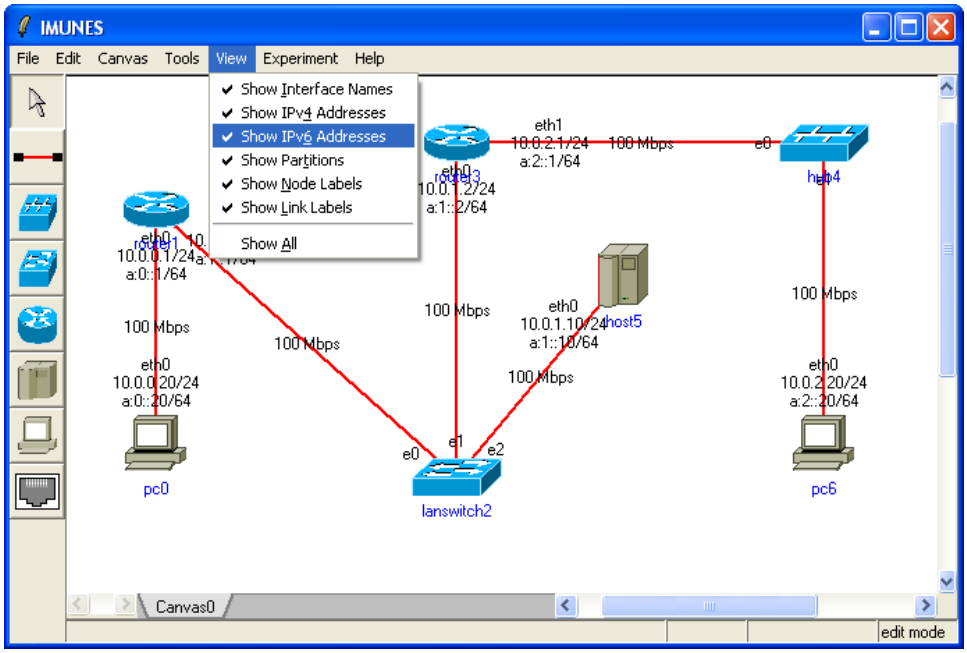


Figure 2.2 GUI of IMUNES

Link layer nodes provide just link layer functionality of transmitting received packets to all exit ports (hub) or just to the port where the packet destination is (switch). There is also a interface node that is used for the interaction of synthetic network with the real one. This node is also a link layer node. Link layer nodes do not pass the packet through TCP/IP stack.

Network layer nodes are the ones capable of acting as a packet source and destination. They have full TCP/IP stack functionality. In IMUNES three predefined network level nodes exist: PC, host and router. The difference between various network layer nodes is in the booting process and configuration.

Routers are predefined to route the packets. For routing the packets the router can use static or dynamic routing through routing models quagga [36] i xorp [49]. Default configuration for the router consists of using quagga model with rip routing.

Host and PC do not forward packets and use static routing by default. The host starts additional services (like telnet) during the booting process, while the PC starts clean. This is the predefined version, but each user can specify what he wants to start on the machine during the emulated booting process.

On any network layer node in IMUNES after starting the experiment an UNIX shell can be opened and any of the standard applications that are available for the hosting machine can be started.

In IMUNES only one link between any two nodes can be created. All links are presumed to be operating in full duplex mode. In addition to creating nodes and links in IMUNES their properties can be changed through GUI. Different routing models, queuing disciplines, services that will be started during the execution of the experiment, as well as bandwidth of the links, can be specified through GUI.

2.2 IMUNES properties

IMUNES is an acronym for the Integrated Multiprotocol Network Emulator/Simulator. The multi-protocol part of the name is justified by the fact that IMUNES provides the support for different routing protocols, as well as providing the support for both IPv4 and IPv6 traffic.

Compared to virtualization used in other machine emulators [42], [17] IMUNES offers higher scalability, being able to emulate 10s to 100s of network level nodes on only one commodity PC. This kind of scalability is a result of lightweight virtualization used in IMUNES, described in next section.

Fidelity of IMUNES is another important issue. In IMUNES all the network layer nodes use real kernel calls that are processed in the same way they are processed on a real machine. Furthermore, this results in IMUNES being capable of using any standard UNIX application (Figure 2.3).

Because IMUNES uses kernel level objects, it is strongly dependent on the kernel, i.e. FreeBSD kernel [12] The version of IMUNES that is used as a base for distributed version is built on top of a FreeBSD 4.11 kernel.

2.3 IMUNES implementation

GUI and the management unit of IMUNES are written in the Tcl/Tk language [6].

For emulation purposes IMUNES maps the GUI objects, nodes and links, to kernel level objects. Kernel level objects used for emulation are virtual machines and kernel level interconnection nodes.

Links and link layer nodes are designed as netgraph nodes, where the netgraph is an interconnection module and a standard part of FreeBSD kernel. Some of the netgraph nodes existed before, like ng_hub that represents hub in IMUNES or ng_bridge

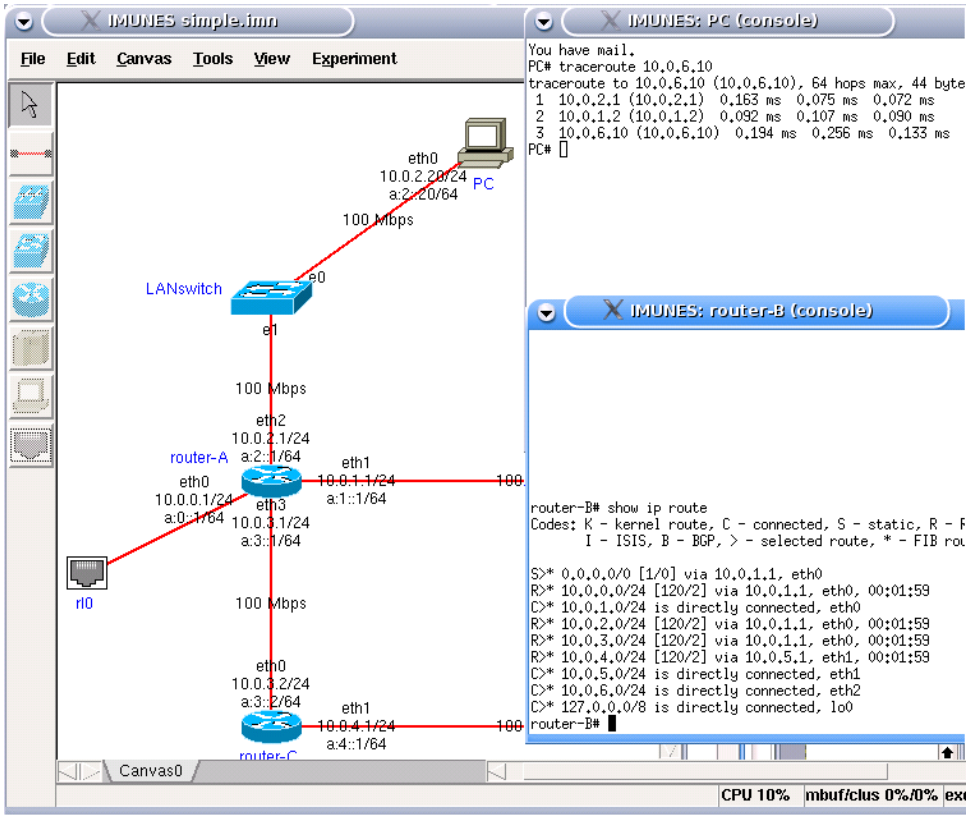


Figure 2.3 Usage of standard UNIX application within emulated nodes

that represents LAN switch in IMUNES. For emulation of links, a new type of node was designed. This type of netgraph node supports all the link properties like bandwidth, delay, BER as well as queueing policy of the interface connected to the node. The netgraph node representing the link is named ng_pipe node. A detailed description of all netgraph nodes listed here is available through man pages.

Network layer nodes are implemented as virtual machines. Virtual machine support is custom designed and presents the heart of IMUNES. Instead of using the usual approach of full virtualization [42] IMUNES uses the lightweight virtualization method. Full virtualization reduces the scalability of virtual machines, since for each emulated packet the user space emulator has to pass the packet to kernel by packet data copying and context switching. By using lightweight virtualizaton, in IMUNES, all the virtual machines share the same kernel except from network stack and some other variables that became private for each virtual machine. The difference between these two types of virtualization is presented in Figure 2.4.

Full virtualization has benefits, such as that heterogenous operating systems can be simulated, but because scalability in IMUNES is more important than different operating systems support, the lightweight virtualization approach is preferred. The virtualization support for IMUNES is designed as a kernel patch, so it is kernel dependant. For each new version of the kernel a new patch has to be created. Manipulating with virtual machines is done with a stand-alone application vimage. A detailed description is available through man pages of vimage.

Emulation of all the links and nodes in kernel space removes the unnecessary packet copying while keeping the realistic approach of using the real operating system code in a packet level emulation. All the properties mentioned, like scalability and fidelity, are the result of IMUNES implementation.

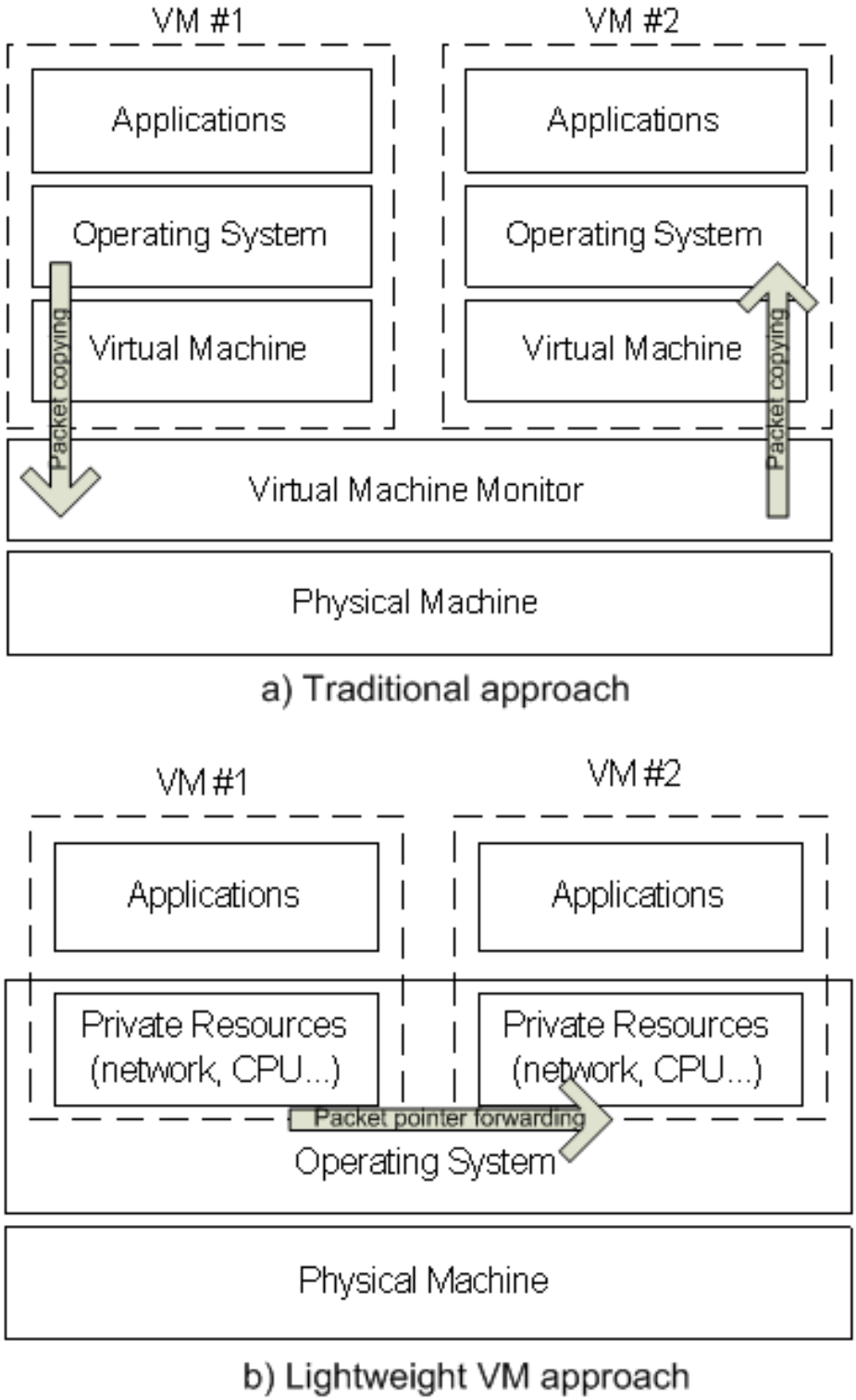


Figure 2.4 Packet passing support

Chapter 3

Topology generation

Although the structure of the Internet is constantly changing, this status may be efficiently described. This chapter describes the evolution of network structures using some standard measurements and Internet-like topology generators. Based on the knowledge about Internet structure an algorithm that creates an Internet-like topology was developed and evaluated.

The evolution of network structures is important since it gives us an idea of the metrics that have been used over the years of describing a network structure. The overview of network structures, presented here, goes from regular latices to scale-free networks, that ultimately describe Internet topology in the best suitable way [25]. The metrics used for describing each network include average path length, clustering coefficient and degree distribution.

Over the years different topology generators emerged because of different interpretations of Internet. These generators produced networks with different underlying structures; some produced random networks [46], some used a hierarchical approach [8] and in the end, after the structure of Internet had been described as scale-free, new topology generators emerged [22], [28], [48].

An algorithm for creating an Internet-like topology [35] was designed and its performance was evaluated. This algorithm combines two different approaches, using the scale-free approach as well as hierarchical approach. In the design of the algorithm it was kept in mind that the algorithm will be used in IMUNES.

3.1 Internet-like topology

In order to describe an Internet-like topology, here is an overview of different networking structures and the metrics used to distinguish properties of the networks.

This overview starts with definitions of metrics used. Then, regular lattices are presented. Further, there is a description of a more recent structure of random networks that are defined with statistical properties. Somewhere in the space between regular lattices and random networks there is a gap that is filled with small-world networks with very specific properties. The most recent discovery in the area of networks is a scale-free network with very distinguishable characteristics. Finally, Internet-like topology is defined as a subclass of scale-free networks.

3.1.1 Metrics

Here the exact definitions of average path length, clustering coefficient and degree distribution are provided.

The average path length is a simple measurement that calculates the average distance between any pair of nodes. Our network is modeled as a graph $\mathbf{\Gamma}(\mathbf{V}, \mathbf{E})$ with the set of vertices $\mathbf{V}$ and the set of edges $\mathbf{E}$. Vertex v_i is an element of $\mathbf{V}$. Edge e_{ij} is in set $\mathbf{E}$ if there exists an edge between vertices v_i and v_j. Since the graph is undirected, e_{ij} is equal to e_{ji}. The path from v_i to v_j is a set of edges of the form of $\mathrm{e}_{ig}...\mathrm{e}_{hj}$. The minimum path is the smallest set of this kind. The path length, d_{ij} is the number of elements in the minimum path set. The average path length for a vertex is the average distance from that node to all the other vertices in graph 3.1. The average path length of the graph is the average path length of each vertex averaged over all the vertices [25]

$$d = \frac{1}{n \cdot (n-1)} \sum_{i \neq j} d_{ij}. \tag{3.1}$$

The clustering coefficient can be defined in two different ways [25]. The clustering coefficient presented here was introduced by Watts and Strogatz [45]. This definition is based on the number of triangles that are connected to a vertex. A triangle is a structure in the graph which occurs with the existence of edges in the form of e_{ij}, e_{jk} and e_{ki}. The clustering coefficient for a vertex v_i is defined as the ratio

$$C_i = \frac{\text{Number of triangles connected to } v_i}{\text{Number of possible triangles on vertex } v_i}. \tag{3.2}$$

The average clustering coefficient is calculated by averaging the vertex clustering coefficient over all the vertices in the graph.

$$C = \frac{1}{n}\sum_{i}^{n} C_i. \tag{3.3}$$

A large clustering coefficient typically means that the value of the clustering coefficient decreases at a rate much slower than $O(1/N)$ where N is the total number of nodes [44].

The last metric considered is the degree distribution metric. The degree of a vertex is simply the number of edges that are connected to that vertex, i.e. the number of neighboring vertices.

3.1.2 Regular lattices

Regular lattices are highly structured networks where each node is connected with a given number of nodes closest to it. A regular lattice is constructed with a given number of nodes and a given number of neighbors. Each node is connected to a $2k$ number of neighbors. The resulting structure is presented in Figure 3.1. This structure is very well defined in the literature in terms of degree distribution, connectivity, clustering coefficient and average path length. When talking about degree distribution, each node has the same degree, so there is no distribution, instead there is a predetermined value of the degree. In terms of measurements of connectivity,

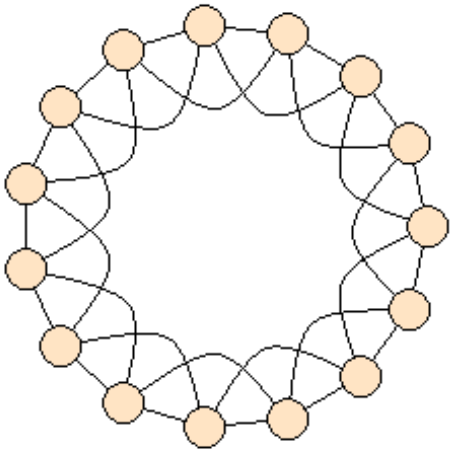


Figure 3.1 Regular lattice

the structure is connected for each $k \geq 1$. The average path length of a regular lattice increases proportionally with the number of nodes in the network. The clustering coefficient of a regular lattice is constant, so it does not change with the growth of the network.

3.1.3 Random networks

Random graphs are mathematical structures describing random networks. They were introduced by Erdös and Rènyi in 1959 (Figure 3.2). A random graph can be created by placing vertices on a plane and then randomly connecting pairs of vertices. In random graphs, there is a well-defined threshold value of edges after which the resulting graph becomes connected [25].

The average path length of a fully connected random graph is small, i.e. it is proportional with $\log(N)$ where N is the number of nodes in random graph. Degree

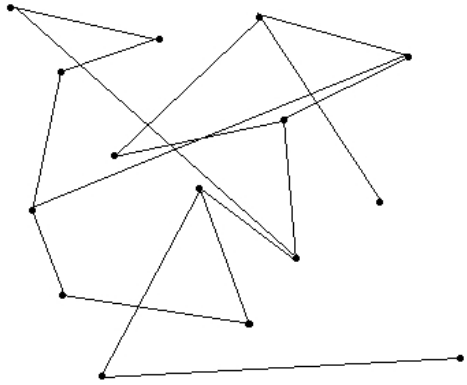


Figure 3.2 A random network

distribution of a random graph follows the Poisson distribution (Figure 3.3). The clustering coefficient, on the other hand, tends to zero when the size of the network tends to infinity.

3.1.4 Small-world networks

Before the introduction of small-world and scale-free networks, the two most widely used graphs were regular lattices and random graphs.

Small-world graphs (Figure 3.4) lie somewhere in between random graphs and regular lattices [44]. Namely, small-world graphs have a small average path length, comparable to that of random graphs, while their clustering coefficient approaches some small, but non-zero, value like in regular lattices.

The most popular algorithm for generating small-world networks starts with a regular lattice. Edges of a regular lattice are rewired with certain probability. As this rewiring probability increases, the graph transits the phases from a regular lattice, through the small-world model and ends up as a random graph. The small-world

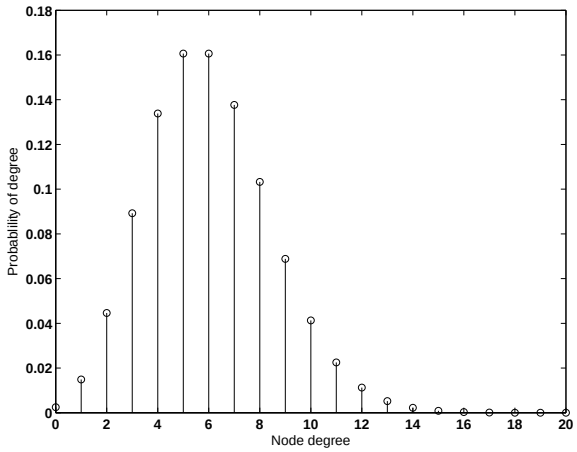


Figure 3.3 Poisson distribution

regime is the narrow area characterized by a small average path length and yet still a large clustering coefficient.

In small-world networks the average path length increases as log(n) [44]. The clustering coefficient tends to a finite small value with the increase in size of the network. Degree distribution of a small-world network is still Poisson, the same as in random networks.

3.1.5 Scale-Free networks

The scale-free property of the Internet was first observed in 1999 by the Faloutsos brothers [10]. The scale-free property actually means that the degree distribution follows a power law in the form of

$$P(k) \sim k^{-\alpha}. \tag{3.4}$$

An example of a scale-free network is shown in Figure 3.5. The scale-free property of networks is sometimes referred to as the 80-20 law or the rich-get-richer-law, meaning that a small percentage of nodes dominate most of the network connections.

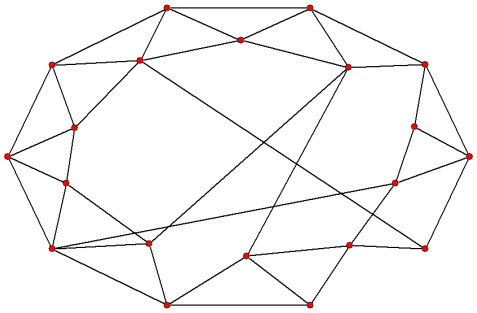


Figure 3.4 A small-world network

The nodes with a very large number of neighbors are called hub nodes. Hub nodes represent a minority in the network, and yet they control the majority of network connections.

The power law degree distribution leads to two interesting properties of the network [4], [25]: the network is very resilient to random failures, but very fragile in the presence of targeted attacks on the hubs and viruses will always spread and persist in the network.

Barabási [4] provided an explanation for the origin of scale-free networks based on the mechanisms of preferential attachment and network growth. Preferential attachment refers to the concept that vertices with higher degrees are more likely to obtain new connections, where the probability of gaining new connections increases linearly with the current degree of the node. As a network grows, new nodes are continuously added to the network and connect to existing nodes using preferential attachment [4].

Although this is the most common approach to generating scale-free networks, these networks can be obtained by using different generation models as described in [25].

Scale-free networks have a small average path length and a finite clustering coef-

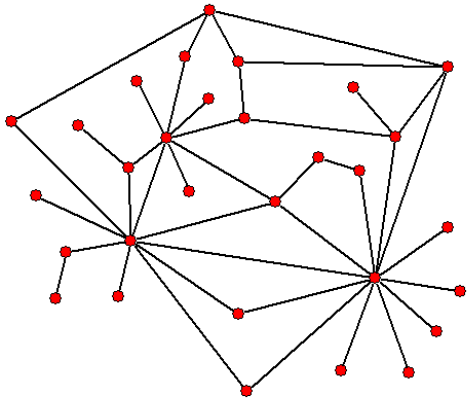


Figure 3.5 A scale-free network

ficient.

3.1.6 Internet-like topologies

Today, the Internet is most often described as a scale-free network [25]. But in addition to the scale-free property the Internet also has a hierarchical structure.

The hierarchical structure of the Internet consists of three different layers: LANs, stubs and transit domains [8]. Based on hierarchical models, nodes with the most connections are placed at the edges of the network. However, modeling the Interent topology using just this hierarchical approach neglects the fact that the Internet shows global behavior that goes beyond this hierarchy. In the large-scale mapping of the Internet in 1999, [10] it was discovered to have a scale-free property.

Preferential attachment and network growth principles, as described in the previous section, result in scale-free networks [4], but they are not considered to describe the Internet topology well [1]. This is because preferential attachment and network growth model the network in such a way that the hub nodes are always the oldest

nodes and mainly lie in the center of the network.

3.2 Internet topology generators

Three different types of generators are available in literature for modeling Internet-like topologies:

- Random topology generators
- Hierarchical generators
- Scale-free generators

The first Internet-like topology generator was designed by Waksman in the late 80's [46]. This model is a flat model since it all the nodes are at the same level. The nodes are interconnected by assuming that the probability of a link decreases exponentially with the distance between two nodes. For a long time, this model was considered best, despite the fact that the underlying graph was basically random.

The next generation of network topology generators, introduced in the mid 90's, was characterized by hierarchical structures [8]. The Internet was divided into three different hierarchical levels:

- LANs - representing a local area network.
- Stubs - representing a network of interconnected LAN-s. Here traffic originates and ends within a single domain.
- Transit - representing backbone nodes. Here traffic crosses the transit domain when going from one Stub to another.

Two different models for generating Internet-like topologies that fall into the category of hierarchical network generators are the Trans-Stub model and the Tiers model [8].

The Trans-Stub model only works with one type of nodes - routers - and provides support for Stub and Transit levels. Tiers uses a minimum spanning tree to create all three levels of hierarchy and assigns weight to the links. It was shown in [28] that neither the Waxman topology generator nor the hierarchical models follow power laws.

The most recent class of network topology generators emerged at the beginning of this decade, influenced by the discovery of Internet topologies following a power law [22], [48], [39] and [28]. These Internet-like topology generators provide a way of generating networks on two different levels, the autonomous systems level [39], [48] and the router level [22], [28]. Both of these models result in topologies with just one type of nodes on one hierarchical level. Autonomous systems (AS) [16] level topologies represent topologies where one node represents one AS. Some scale-free generators offer the possibility to analyze the resulting networks by comparing the network parameters generated with real Internet data [22]. In order to obtain power laws, some of the algorithms may yield unconnected networks [28], which are later discarded, and the process is repeated until a connected network is obtained. Some generators take into account the physical placement of the nodes [39], providing another similarity with real Internet topologies.

3.3 Our topology generator

Here is an overview of our topology generator and an evaluation of it based on the metrics described in the previous section.

In order to generate an Internet topology, an approach of combining the hierarchical approach [8] with power law models [28] was used. The resulting network topology is presented in Figure 3.6. From the Figure it can be observed that there are three different hierarchical layers. The process of creating the topology consists

of three phases. In the first phase assignment of target degrees to all the nodes in the network takes place. Next, a small-world network is created as the core topology. This network consists only of routers, and it consists of 20% of all nodes. In the last phase local area networks are created and connected to the core routers. In assigning degrees to the nodes the target degrees are calculated to be as close as possible to the values assigned in the first step.

Our algorithm is evaluated based on the measures of average path length, clustering coefficient and degree distribution. Based on obtained values, the resulting topology matches the standards of scale-free topology.

3.3.1 Algorithm

The target network is hierarchical and has three different layers. On the first layer, routers are interconnected into a small-world network. The core router network, thus, has a small average path length and a large clustering coefficient. On the second layer, hub nodes, which correspond to real world hubs, are connected so that they are responsible only for local nodes. On the last layer, PC representees are connected to the hub nodes.

The pseudo-code of the algorithm is presented in Figure 3.7. The parameters of the algorithm are:

- n - the number of nodes in the network,
- *alpha* - the degree distribution exponent,
- k - the average small-world degree of a router,
- p - the small-world rewiring probability.

The algorithm starts by generating a power law distribution with a specified power law exponent using a method similar to that described in [28]. After creating a

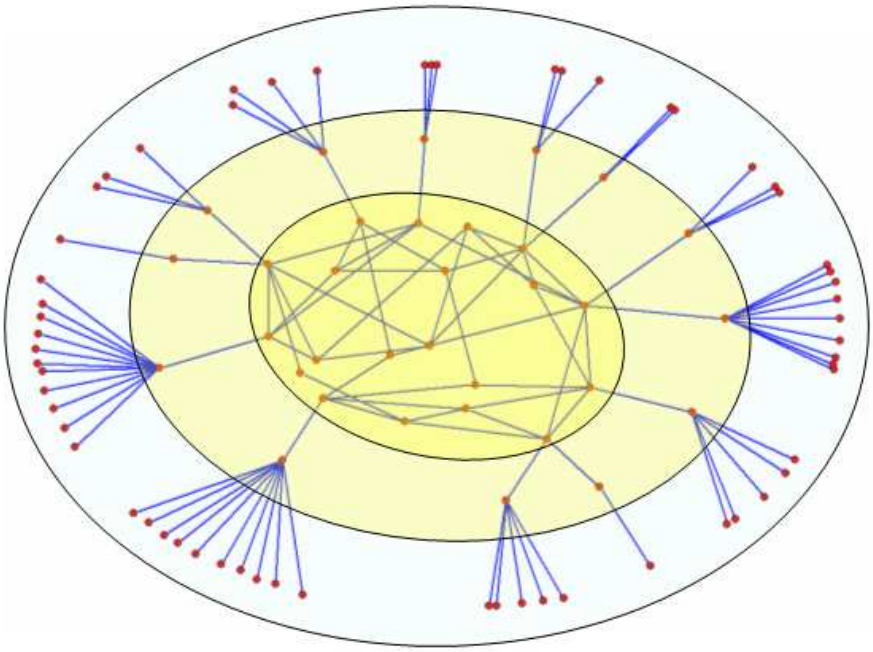


Figure 3.6 The generated network

power law degree distribution with parameter *alpha*, the process continues by creating a small-world network for the core, where 20% of the nodes are routers with an average small-world degree k, and with rewiring probability p. Once a small-world core network is created, the corresponding node degrees are mapped to the pre-calculated degrees for the whole network. The mapping takes into account the target degree of the core network as opposed to the current degree, leaving free connections for hub nodes. Next, a degree from the rest of the degree distribution pool is assigned to the hub nodes. These nodes are then connected to routers and to a certain number of PCs corresponding to the degree specified. The mapping between the resulting and specified degrees is not exact, i.e. the resulting degrees are allowed to be different from the specified ones.

3.3.2 Results

The resulting network topologies and the corresponding degree distributions for networks with 200, 500 and 1000 nodes are presented in Figures 3.9-3.11. The resulting power law exponent differs from the specified one, since the algorithm does not support exact mapping between the obtained degree distribution and the artificially created degree distribution. As observed, the resulting degree distribution of the algorithm follows a power law distribution, so the creation of a small-world core network does not significantly influence the resulting distribution shape, although the resulting power law exponent changes from its initially specified value.

The curve representing the resulting degree exponent of the differently sized networks is presented in Figure 3.8. Our results are compared to those published in [22]. Those measurements are not repeated here, but instead the given threshold and target values are used. One can see that, even for moderately sized networks, the obtained degree exponent values are within 2%.

```
power_law(maxDegree = n, alfa)

for (i in 1 to n):
  degree(i) = power_law.getDegree(random())

smallworld(routers, k, p)
pcs = Count(elements with degree==1)

hubs = n - routers - pcs
for (i in 1 to routers):
   map_smallworld_to_degree
   leave free connections for hubs

is_hub=true
for (i in k to n):
  if is_hub:
    map degree
    connect to random free router
    nPcs = hub.free_connections
    is_hub = false
  else:
    connect to hub
    nPcs--
    if nPcs = 0 is_hub=true
```

Figure 3.7 Pseudo-code of the proposed algorithm

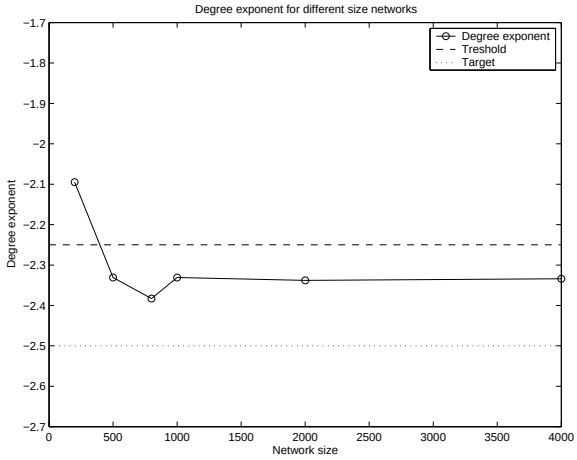


Figure 3.8 The degree distribution exponent for networks of different sizes

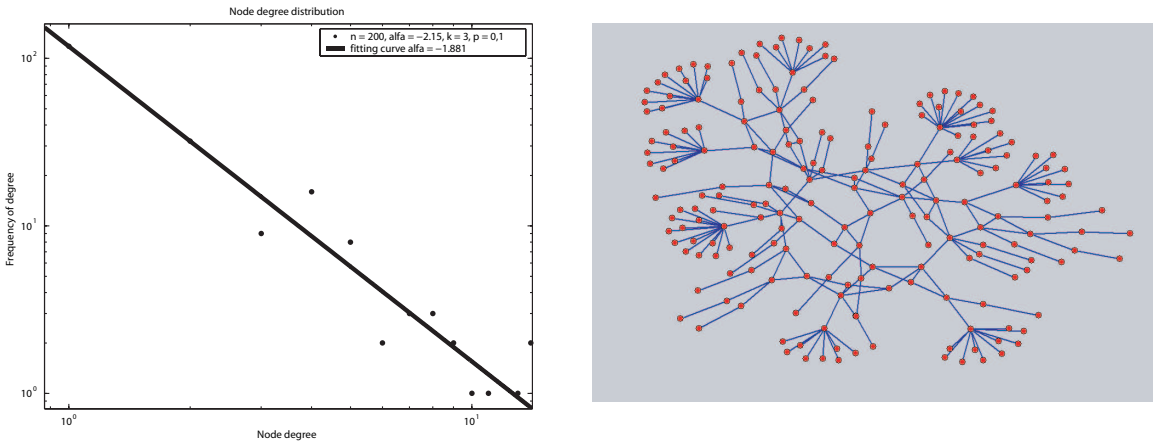


Figure 3.9 Network topology with 200 nodes

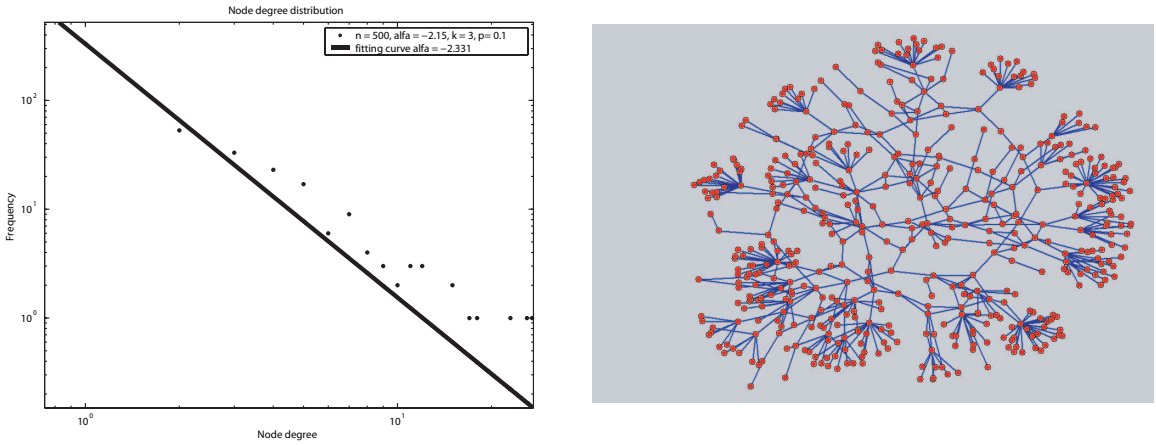


Figure 3.10 Network topology with 500 nodes

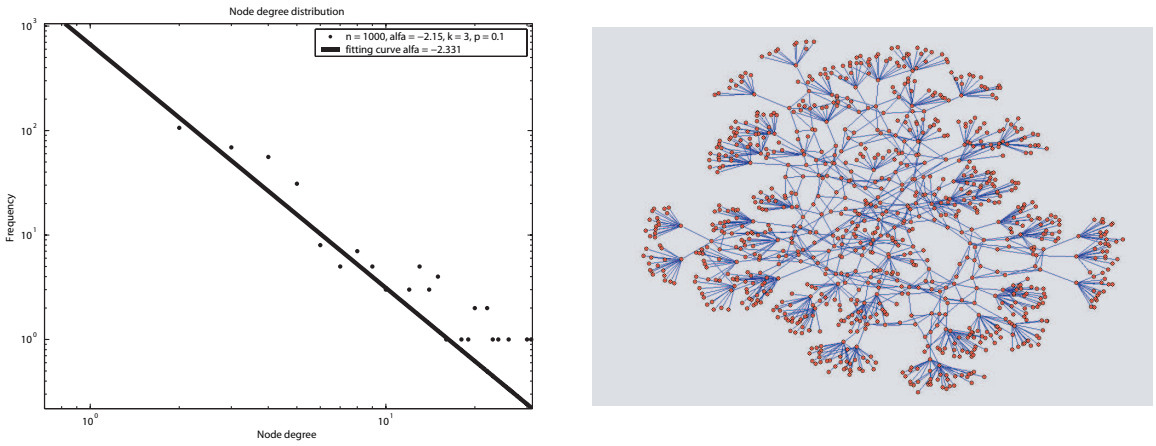


Figure 3.11 Network topology with 1000 nodes

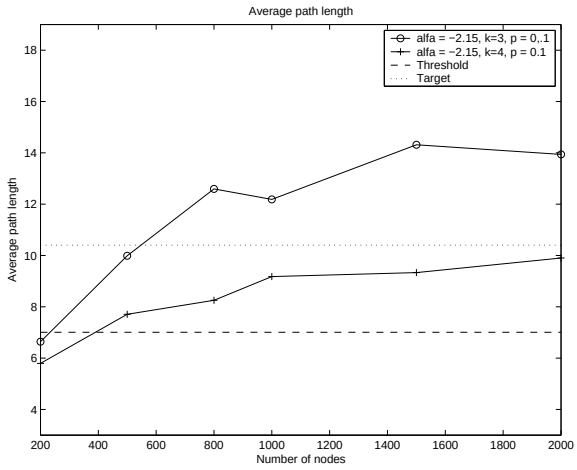


Figure 3.12 The average path length

In Figure 3.12, the average path length as a function of the number of nodes in the network is presented. The average path length is influenced by the selection of parameter k. Since parameter k influences the small-world core network, a conclusion can be drawn that the average path length is highly dependant on the core network's properties. The target and threshold values correspond to those presented in [22].

From Figure 3.13, it is evident the clustering coefficient remains relatively constant for different sizes of the network as well as for different values of k. The clustering coefficient remains the same regardless of network size, which is the behavior of Internet-like topologies as well.

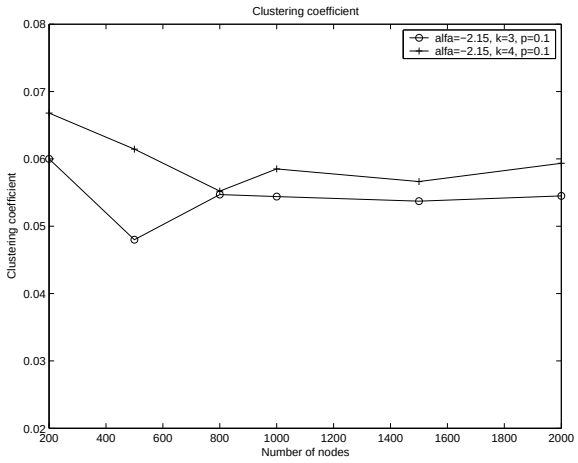


Figure 3.13 The clustering coefficient

Chapter 4

Topology partitioning

Dividing a network topology into parts is necessary for distributed emulation. All available resources that are distributed must be mapped to the network topology without redundancy. There are various techniques available when dividing a topology into parts. The focus here is on graph partitioning, although some other methods also exist in the literature [37]. Here, the road map for the rest of this chapter is presented.

First, a description of graph partitioning problem is given. Graph partitioning is the process of finding disjoint sets of vertices, i.e. partitions with some constrains. The weight of the partition represents the sum of the weights of all vertices belonging to that partition. All the partitions must have equal weights and the total weight of all the edges connecting vertices from different partitions must be minimum. With this optimization criteria the graph partitioning problem is an NP-complete problem (for definition of NP-complete problem refer to [9]), so the proposed solutions are heuristical. Good heuristics have been developed to deal with this NP-hard graph partitioning problem. The most common include spectral graph partitioning [31], genetic algorithms [23], greedy heuristics [18] and multilevel approach [19].

Second, there is conversion of network topology into a graph so graph partitioning methods can be used. The usage of graph partitioning approach is not new in network simulation. As presented in [50] there have been other attempts of implementing the graph partitioning method in distributed network simulation. Our attention is focused on converting IMUNES topology into a suitable graph. There is also a description of our modifications to the algorithms used.

Third, the results are presented and the usage of two different graph partitioning

methods on IMUNES is discussed. The chosen algorithms are referred to and the applicability of both of them is discussed.

Finally, further research topics for this area are given. The usage of graph partitioning methods is not the only option in dividing a network topology. Recently, a new kind of problem was introduced that addresses the issue of mapping a network topology into a testbed topology [37]. This new method has been adopted in network simulators like [15] and [5].

In the distribution of simulation the goal is to maximize the usability of available network resources. In order to do so, nodes which are deployed on different machines are disjoint, i.e. one node is simulated on only one machine. This way there is also a reduction in the cost of overall synchronization between multiple instances of the same simulated node. Graph partitioning algorithms are a good tool for creating subsets of nodes that meet these demands.

4.1 Graph partitioning

The graph is defined with the set of vertices $\mathbf{V}$=v_1, v_2, ..., v_n and set of edges $\mathbf{E}$=e_1,e_2, ..., e_m. Any edge in the graph is defined as a pair of vertices that it connects. A graph is directed if the pair of vertices defining an edge is ordered. The graph can also be weighted, and then it is defined as $\Gamma(\mathbf{V}, \mathbf{E}, \mathbf{w^v}, \mathbf{w^e})$. $\mathbf{w^v}$ is a function that assigns weight to each vertex, i.e. $\mathbf{w^v} : \mathbf{V} \rightarrow \mathbf{R_0^+}$. $\mathbf{w^e}$ assigns weight to each edge, i.e. $\mathbf{w^e} : \mathbf{E} \rightarrow \mathbf{R_0^+}$, where $\mathbf{R_0^+}$ is a set of positive real numbers including zero.

Clustering of a graph is defined as finding subsets of the set of vertices $\mathbf{V_1}, \mathbf{V_2}, ...\mathbf{V_n}$ such that $\bigcup \mathbf{V_i} = \mathbf{V}$. A subset of vertices $\mathbf{V_i}$ is also called a cluster.

The graph partitioning problem can be viewed as a special case of the clustering problem, where clusters are disjoint. The problem of n-way partitioning is as follows. The set of vertices $\mathbf{V}$ is divided into disjoint sets $\mathbf{V_1}, \mathbf{V_2}, ..., \mathbf{V_n}$ with the following

properties:

1. $\bigcup \mathbf{V_i} = \mathbf{V}$
2. $\mathbf{V_i} \bigcap \mathbf{V_j} = \emptyset$, for i $\neq$ j

If these conditions are fulfilled clusters can also be called partitions.

A subset of $\mathbf{E}$, called $\mathbf{E_0}$ contains all the edges between vertices that belong to different clusters. In other words, set $\mathbf{E_0}$ represents the edge-cut of the partitioned graph.

The n-way graph partitioning problem has two objectives. The first objective is to find the partitions with equal loads, i.e.

$$\sum_{v \in \mathbf{V_i}} \mathbf{w^v}(\mathbf{v}) = \sum_{u \in \mathbf{V_j}} \mathbf{w^v}(\mathbf{u}) \tag{4.1}$$

The second objective is to find such partitions for which the sum of the weights of all edges in $\mathbf{E_0}$ is minimized, i.e.

$$\min(\sum_{e \in \mathbf{E_0}} \mathbf{w^e}(\mathbf{e})) \tag{4.2}$$

The problem of partitioning graphs is a NP-complete optimization problem so heuristic algorithms are used to find suboptimal solutions [2]. The most common approach is to solve the problem for 2-way partitioning (so called bisections) and then solving k-way partitioning by recursively creating bisections [19].

Algorithms for graph partitioning rely on different techniques. Techniques which differ with respect to the approach used include spectral methods [31], multilevel techniques [19] or genetic algorithms [23]. Techniques which use a similar approach but differ with respect to implementation include Kernighan-Lin heuristic [2], Fiduccia-Mattheyses heuristic [11]; multilevel heuristic using random matching or heavy edge matching [19]. The performance of two different algorithms was measured: greedy

graph partitioning algorithm [18] and multilevel graph partitioning [19]. The considered algorithms present the fastest graph partitioning methods according to [19]. The description of the algorithms is presented in the following subsections.

4.1.1 Greedy graph partitioning algorithm

This algorithm was introduced by Ciarlet and Lamour [18]. The Greedy partitioning heuristic is as follows. Clusters are subsequently created by iteratively adding nodes for as long as the total weight of the cluster remains less than the previously defined threshold. Nodes included in a cluster are further called marked nodes. Each cluster initially consists of a single node, referred to as the starting point. In each iteration following it, all the unmarked nodes adjacent to the cluster are added if the resulting weight of the cluster is within the allowed bound (less than threshold). If this is not the case, a tie-break strategy is applied. This process is repeated subsequently to create $k-1$ cluster. The k-th cluster is created by simply marking all the remaining nodes. This algorithm deeply depends on the choice of the starting point of each cluster. Those nodes which have the least number of adjacent nodes that are unmarked, i.e. not yet included in any other cluster are considered good.

4.1.2 Multilevel graph partitioning algorithm

The multilevel graph partitioning algorithm was introduced by Karypis and Kumar [19]. This algorithm is called multilevel since it iteratively reduces the number of vertices in the graph. There are three main phases that the algorithm goes through: coarsening phase, partitioning phase and uncoarsening phase.

The coarsening phase is the phase of reducing the number of vertices. In this phase new graphs are iteratively created. Each new graph has a corresponding level. The graph of a level n is constructed by matching two adjacent vertices of the previous level

graph and presenting them with one new vertex in the n+1 level graph. The vertices that were not matched, but have all the neighboring vertices matched, are simply added to the graph. The matching techniques described by Kaypis and Kumar include random matching (for any vertex that is not matched, if an unmatched neighbor is found then they will be matched), heavy edge matching (edges with the highest weight are found and the vertices they connect are matched), light edge matching (edges with the lowest weight are found and the vertices they connect are matched) and heavy clique matching (the matching is performed by the edge density, the sets of vertices that are most highly connected are presented with one vertex on the next level). The matching technique used is heavy edge matching.

The partitioning phase occurs when the graph is completely coarsened, i.e. the number of vertices in the graph is fully reduced. As the partitioning algorithm, greedy graph partitioning is used, but on a smaller number of vertices and edges, therefore reducing the partitioning complexity.

Uncoarsening is the process of restoring the original graph. Here the partitions from the graph on level n are transferred to the graph on level n-1 and refinement techniques are used to improve these partitions, on each stage of the way. The whole algorithm is presented in Figure 4.1.

Implementation of this algorithm is publicly available as a part of METIS program suite [24], and is often introduced as the fastest graph partitioning algorithm [19].

4.2 Application of graph partitioning methods to IMUNES

For applying graph partitioning methods to IMUNES topology it was necessary to define transformation from a network topology into a graph and add some IMUNES specific steps to the partitioning algorithms [32]. In this section there is an overview of IMUNES specific details that lead to the implementation of graph partitioning

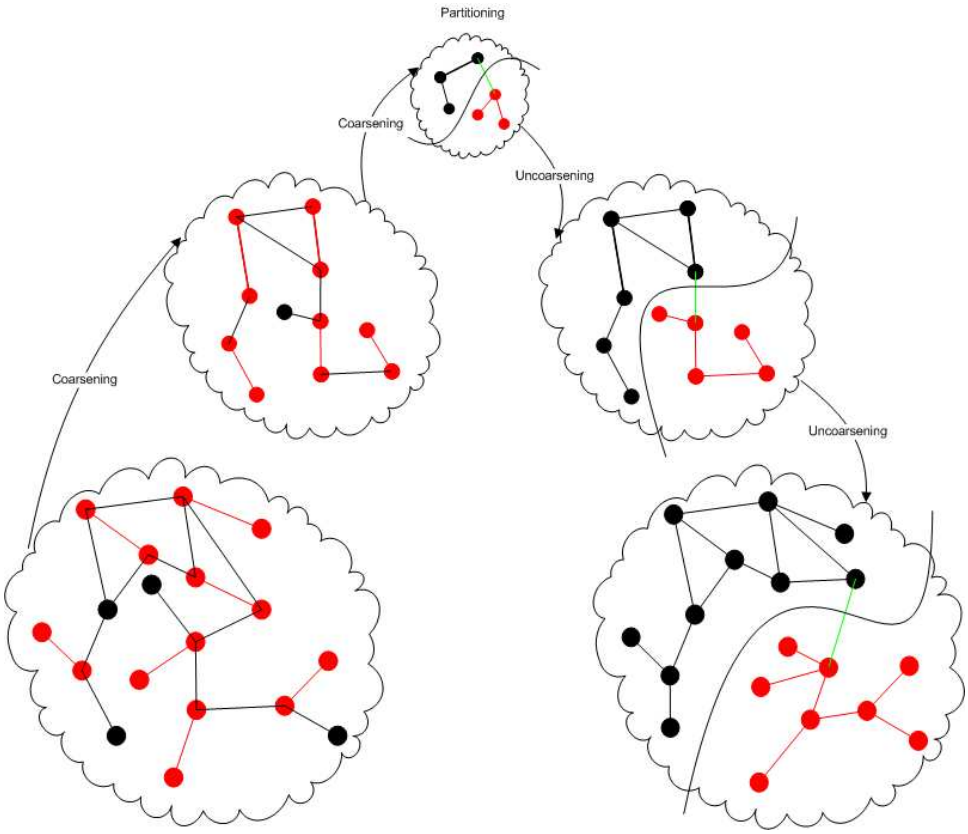


Figure 4.1 METIS phases

algorithms.

4.2.1 Transformation of IMUNES topology into a graph

The description of IMUNES is in Chapter 2, but here some basic facts are revised. IMUNES has two types of construction units: nodes and links, so any simulated network is completely defined with two sets: a set of nodes and a set of links. A weighted graph is constructed: $\Gamma(\mathbf{V}, \mathbf{E}, \mathbf{w^v}, \mathbf{w^e})$, where $\mathbf{V}$ represents the set of vertices corresponding to the set of nodes in the simulation and $\mathbf{E}$ represents the set of edges corresponding to the set of links in IMUNES topology. Function $\mathbf{w^v}$ assigns weight to vertices of a graph and is defined as $\mathbf{w^v} : \mathbf{V} \rightarrow \mathbf{R_0^+}$. Function $\mathbf{w^e}$ assigns weight to every edge of a graph and is defined as $\mathbf{w^e} : \mathbf{E} \rightarrow \mathbf{R_0^+}$.

Nodes in IMUNES are classified into nodes with IP stack and nodes without IP stack. Nodes with IP stack use more resources than those without it.

The nodes with IP stack are: router, PC and host. Routers perform the function of routing the packets in the simulation, and $\mathbf{w^v}(\text{router}) = \mathbf{w_r^v}$. PCs represent the client machines and have assigned weight $\mathbf{w^v}(\text{PC}) = \mathbf{w_p^v}$. Hosts, the most demanding elements, represent the servers, which run all the desired applications. The weight of a host is denoted as $\mathbf{w^v}(\text{host}) = \mathbf{w_h^v}$.

Nodes without IP stack are: hub, LAN switch and physical interface. LAN switches are used for distributing packets to destinations on a local network segment. The weight of a hub and LAN switch is approximately the same, so by definition $\mathbf{w^v}(\text{LAN switch}) = \mathbf{w^v}(\text{hub}) = \mathbf{w_{ll}^v}$. A Physical interface is the emulation interface and is a real physical interface on the machine. Based on an assumption that it does not put any additional weight on the machine we define $\mathbf{w^v}(\text{physical interface}) = 0$.

Adding a node to the set of nodes simulated on one machine increases the load on that machine. This increase in load is symbolically represented by the node weight.

Because the load of each simulated node in IMUNES depends on many factors, there is no fixed number for each weight. The weight is an experiment specific measure and it should be defined per each experiment individually. Currently all nodes of the same type have the same weight. The host has the maximum weight, and the LAN switch and hub minimum. Therefore adding a host would increase the overall load of the machine more than adding a LAN switch or hub to the same machine.

Different types of links can be simulated in IMUNES by setting link parameters to various values. These parameters include: bandwidth, delay, BER and duplicate. For defining the weight of the link only the bandwidth parameter was used. Therefore, $\mathbf{w^e}$ is defined with a function as $\mathbf{w^v}(\text{link}) = \text{link bandwidth in Mbps}$. Links connecting physical interface to a node have maximum weight.

The example of transformation from IMUNES network topology into a weighted graph is displayed in Figures 4.2 and 4.3. The process of creating the graph from the network topology is demonstrated on the elements from IMUNES simulator, but the same process can be used with any other simulator.

4.2.2 Preprocessing steps

Before starting the graph partitioning algorithms on the transformed IMUNES topology there are some additional actions that have to be done having in mind that the constructed graph is in fact a network topology and that the partitioned topology needs to fit it to another topology that has limited resources.

Based on the fact a network topology is in question, it was possible to combine some nodes together to form only one node in a graph. This includes combining physical interface with their adjacent node, as well as combining a local area network into only one node.

An additional constraint in our case is the equipment on which the simulation

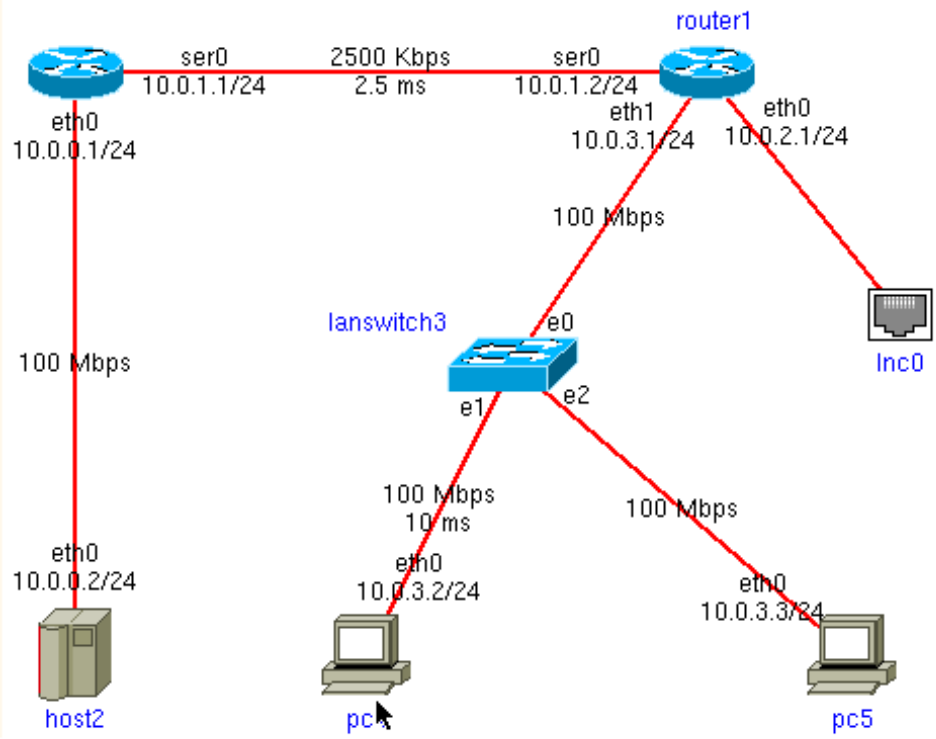


Figure 4.2 Simple IMUNES topology

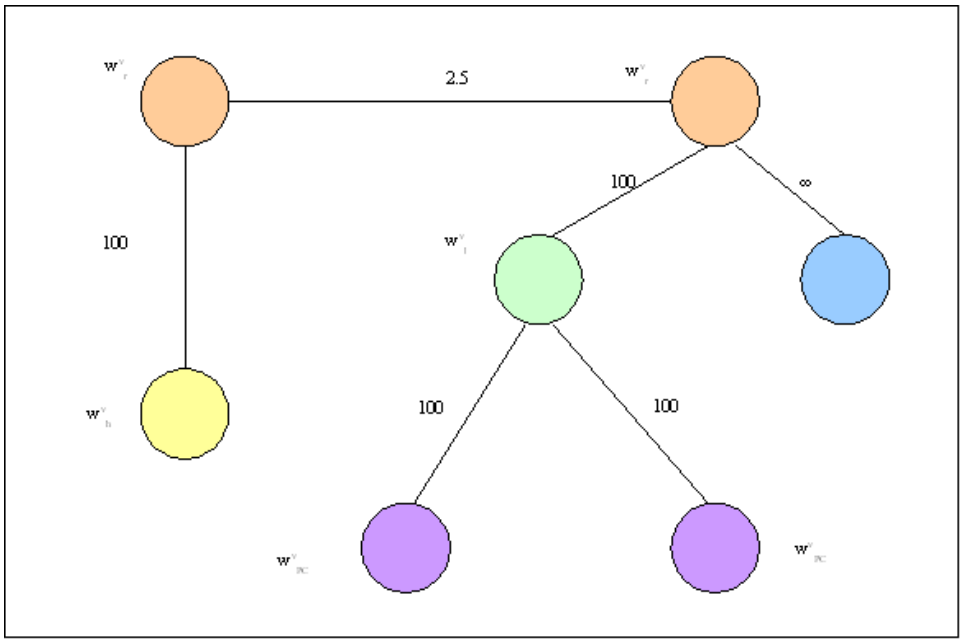


Figure 4.3 Corresponding graph

is deployed. The number of end partitions is equal to the number of machines on which the simulation is run. The bandwidth available between machines dictates the maximum value of the edge-cut (the edge-cut must be lower than the total bandwidth available). There are preprocessing steps which coarsen the graph on the edges that require more bandwidth than available. This process is recursive and ensures that every edge in the final graph can be simulated with the available bandwidth.

4.2.3 Postprocessing steps

Postprocessing steps are introduced to improve original partitions obtained by the greedy graph partitioning algorithm and the multilevel graph partitioning algorithm. As a refinement technique Fidducia-Mattheyses heuristic [11] was used, but only on the vertices with neighbors in different partitions, i.e. boundary vertices. For each vertex in the boundary area the maximum gain and the maximum gain cluster is calculated. Gain of a vertex v for cluster i is equal to the reduction of edge cut achieved by moving vertex v to cluster i. The maximum gain cluster for vertex v is the cluster for which this gain of vertex v is maximum. The process iteratively moves vertices from the original cluster to the maximum gain cluster. In each pass of the iteration, the vertex first to be moved is the vertex with the maximum gain from the cluster with the maximum weight. Once moved, neither a vertex nor any of its neighbors can be moved again during the same pass. This process is stopped after a previously defined number of iterations. The resulting partitions are the best partitions obtained during the process.

4.3 Results

In order to talk about the results, a definition of metrics is required. In this case it is the efficiency metric, that combines the two optimization goals of the graph

partitioning algorithm. With the defined metrics it is possible to compare the results obtained for two different graph partitioning methods and discuss the results.

4.3.1 Metrics

To define metrics of efficiency, the demands on each algorithm have to be defined as well. The demands are presented in terms of minimizing the edge-cut and maximizing the load balance. The presented metrics are contradictory, i.e. by minimizing the edge-cut, load put on every machine varies a lot (as an extreme example consider the case where one machine runs the whole simulation so there is no edge-cut). As a result, two different efficiency metrics are introduced: edge-cut metric and load balance metric.

For the efficiency metric of an edge-cut the ratio of cumulative weight of all edges which are in $\mathbf{E_0}$ to cumulative weight of all edges in $\mathbf{E}$ is used. The value of this ratio is subtracted from 1, to get the efficiency measure in which the higher values become more preferable. The edge-cut measure represents the bandwidth used for communication between clusters. The formula for the edge-cut measure is:

$$\text{eff}_e = 1 - \frac{\sum_{e \in \mathbf{E_0}} \mathbf{w^e}(\mathbf{e})}{\sum_{e \in \mathbf{E}} \mathbf{w^e}(\mathbf{e})} = \frac{\sum_{e \in \mathbf{E} \setminus \mathbf{E_0}} \mathbf{w^e}(\mathbf{e})}{\sum_{e \in \mathbf{E}} \mathbf{w^e}(\mathbf{e})} \tag{4.3}$$

The value of eff_e is in the range of 0 to 1. The higher the value of edge-cut efficiency the lower is the total bandwidth that connects the clusters. The eff_e can be equal to 1 only in the case of connected topology when all the nodes are simulated on only one machine.

For efficiency metric of load balance, load balance metric is used. The deviation of load for each machine can be computed in the following way [23]:

$$l(i) = \left(\frac{\sum_{v \in \mathbf{V_i}} \mathbf{w^v}(\mathbf{v}) - \sum_{u \in \mathbf{V}} \frac{\mathbf{w^v}(\mathbf{u})}{k}}{\sum_{u \in \mathbf{V}} \frac{\mathbf{w^v}(\mathbf{u})}{k}} \right)^2 \tag{4.4}$$

The lower the value of $l(i)$ the partition is more balanced in the sense that the load of the partition is more similar to the average load per partition. The efficiency of balancing the load is defined in the following way:

$$\text{eff}_l = e^{-\sum_{i=1}^{k} l(i)} \tag{4.5}$$

The same as in the case of edge-cut metric, this metric is also limited to the interval $[0, 1]$. In this case the best result is reachable for perfectly balanced partitions, i.e. the eff_l can be 1.

The total efficiency metric combines both of these metrics. The total efficiency of partitioning is calculated as a weighted combination of the above mentioned efficiencies:

$$\text{eff} = r \cdot \text{eff}_e + (1 - r) \cdot \text{eff}_l \tag{4.6}$$

Factor r determines which objective criteria dominate. When r is equal to 1, the edge-cut efficiency alone represents the total efficiency.

4.3.2 Results

Both algorithms described were tested on randomly generated networks. The networks were sparse and had 22, 33, 44 and 53 nodes. The results for bisection are displayed in Figure 4.4 and 4.5. Partitioning was performed for $k \in \{2, 3, 4, 5\}$, and the multilevel algorithm had higher efficiency for network with differently weighted links.

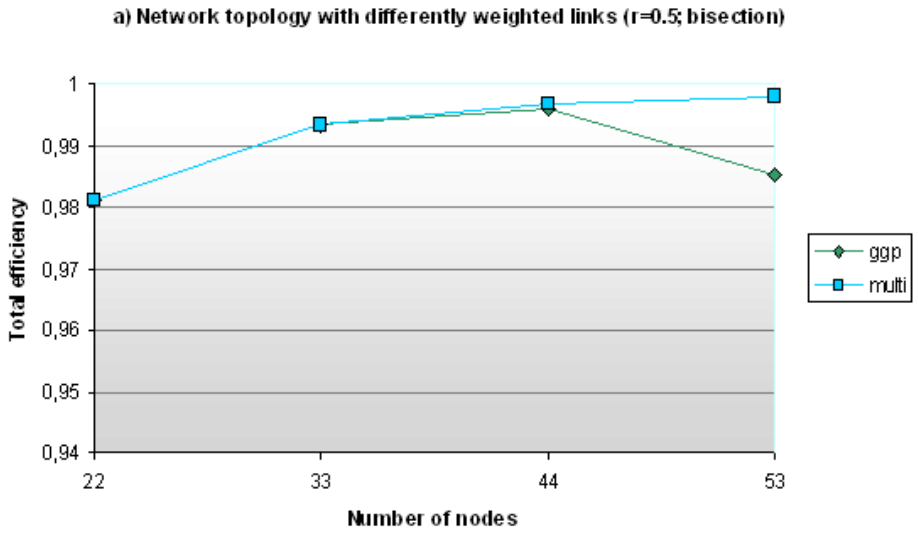


Figure 4.4 Bisection of networks with links of different bandwidths

The multilevel algorithm (multi) showed better graph partitioning results for graphs with a higher number of parts and for larger graphs with differently weighted edges. However, the greedy graph partitioning (ggp) algorithm seems to perform better in the environments with equally weighted links.

However, in our network topology scenarios most of the links have different bandwidths (the local area networks are usually connected with more bandwidth than wide area networks). So, the support for multilevel graph partitioning through METIS software package was implemented in our network simulator.

4.4 Future work

The idea of distributing network simulation over an existing network is not a new one. The benefits of clustering network simulation are various, from increased scalability and functionality to improved simulation capabilities. ModelNet [50], a large

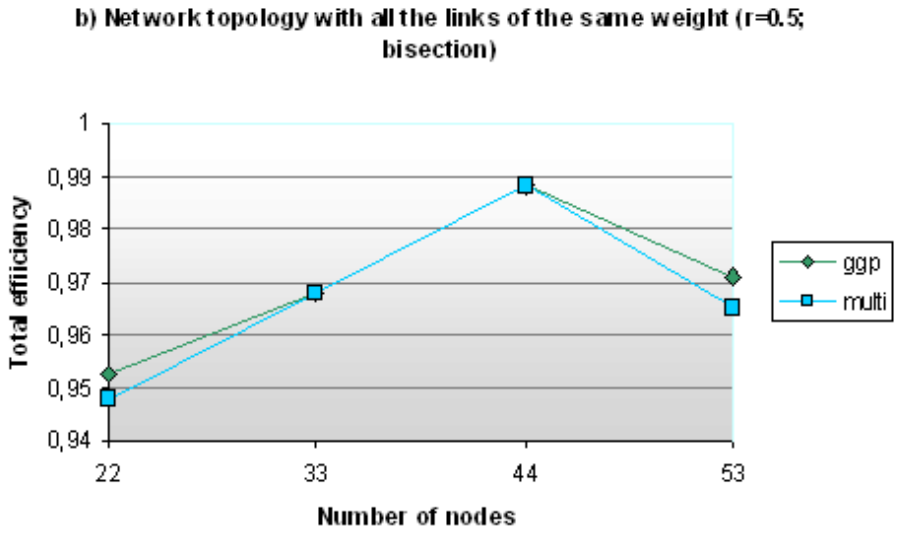


Figure 4.5 Bisection of networks with links of same bandwidth

network emulator, uses the standard graph partitioning methods for clustering network topology. In [50], network topologies are transformed into weighted graphs and the simulation is deployed over an existing topology. Comparing graph partitioning methods to random partitioning significant benefits have been found in clustering the simulation by using standard graph partitioning methods.

In this chapter the procedure of distributing network simulation using the graph partitioning approach was presented. The IMUNES network topology is efficiently transformed into a weighted graph and partitioned using standard graph partitioning methods. The proposed efficiency measure reflects the characteristics of edge cut measure or load balancing measure depending on the value of parameter r. Results indicate that both of the tested algorithms have good properties and can be used for partitioning different types of graphs, i.e. the multilevel technique for graphs with differently weighted links and the greedy graph partitioning method for graphs with equal edge weights. Because of the results usage of multilevel graph partitioning approach is preferable, since in network topology scenarios a variety of different bandwidths can be encountered.

The promising approach presented in [37] is another approach that was not tested in our environment, but because this approach is used in many network simulators [15], [5] the support for this kind of mapping should be implemented in the future.

Chapter 5

Going distributed

5.1 Theoretic background in distributed systems

The trends of globalization, networking and mobility increase the importance of distributed systems. Resource sharing that distributed system provide can be viewed in two major ways, one is in having larger resources available, and the other is in the flexibility of approaching the resources from different locations. A distributed system in those terms provides larger resources that are shared among people and have a greater efficiency.

The distributed system as defined in [14] is a system of interconnected machines that communicate to each other by passing messages. Based on this definition the authors draw the following consequences:

- Concurrency - all the machines work at the same time on the same problem or on different problems.
- No global clock - there is a limit in achieving synchronization because communication is allowed only through the messages.
- Independent failures - all components can fail independently, so the designer must have that in mind when designing a distributed system.

These consequences are the inherited properties of all distributed systems and also indicate the main challenges of distributed systems.

5.1.1 Desired properties

When designing distributed systems there is a list of challenges. Depending on the purpose of our distributed system some of these challenges are more or less important.

Heterogeneity

In the definition of the distributed systems it was mentioned that all the communication is done by passing the messages, but it was not specified what kind of network or what types of machines are cooperating. Ideally, achieving perfect heterogeneity means to be able to pass the messages regardless of the underlaying network or the types of the end machines. To achieve heterogeneity in the past years the term of *middleware* was coined. Middleware is a layer that masks all the differences between underlaying networks and machines and presents a common interface to upper layers. It can also be viewed as a distributed operating system that resides on top of the base operating system of each machine. Middleware will be described in more detail later in this chapter.

Openness

Openness is the property of distributed systems that allows a system to be extended in an arbitrary way, including the addition of hardware or software. In order to make this possible, the distributed system has to have published and well defined key interfaces, a standardized way of accessing the shared resources, and all the components must be carefully tested so that they truly correspond to the interface.

Security

Distributed systems present shared resources and in that sense they have great value for a group of people, so security issues are very important. In meeting security

requirements there are two main challenges: first, to send a message over the network in a secure manner and second, to identify the remote user correctly. The first challenge is met with the usage of cryptography and the second one with authentication. If both of these methods are used, the communication is going through a secure channel. Most of the security threats are resolved with secure channels, but some attacks like denial of service have not been resolved yet.

Scalability

In order to be scalable, a distributed system must be cost effective, extendable and the performance loss by addition of new resources in the system must be controllable. In addition, the developer must take care that the software resources do not run out (IP addresses) and that performance bottlenecks are avoided.

Failure handling

Failures in distributed systems show greater variety that in undistributed systems. For example, it is possible that some of the components fail while other continue to work properly. The failure handling process goes from detecting failures and if possible masking or tolerating failures, if not then recovering from failures. Here redundancy must be mentioned, as a means of tolerating failures.

Transparency

Transparency is a property of accessing and using a distributed system as a whole without a need to identify the system components. This challenge is one of the most important challenges when designing a distributed system. As a result, transparency is classified as:

- Location transparency - the resource location is not known to the user, by

accessing the resource the user does not have to know the physical location of the resources.

- Name transparency - the name of the resource is not directly connected to the location of the resource.
- Access transparency - both local and remote resources are accessed using the same operations.
- Migration or mobility transparency - migration of the resources and clients without affecting the job that is being executed.
- Replication transparency - in the system there might be multiple copies of the resource but this fact is concealed from the user or application programmer.
- Concurrency transparency - multiple processes can use the same resource without interference.
- Failure transparency - if a part of the system fails the whole system does not.
- Scaling transparency - addition of the new resources does not require changes in algorithms or structure.

Access and location transparency are sometimes referred to as network transparency and they present the most important transparencies for our model.

5.1.2 Architectural models

The architectural models of distributed systems simplify the functions of individual components and present their placement and interrelationships. This simplification in terms of process classifies processes as server processes, client processes and peer processes.

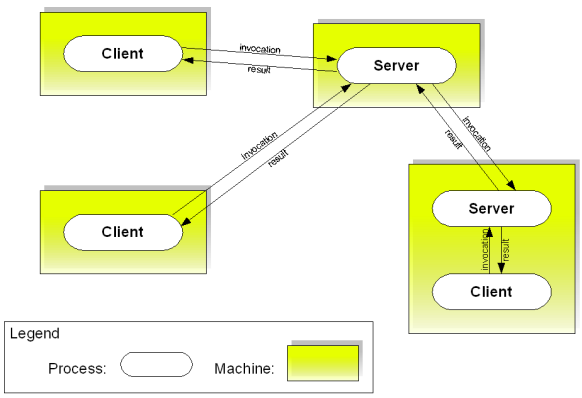


Figure 5.1 Client-server model

The distributed system can be constructed as centralized or decentralized. The centralized solution is the simplest solution, providing easy software update. But at the same time central authority is the most critical part of the system - if it fails, the whole system fails. Distributed or decentralized systems suffer from other types of problems, like the increase of the overall traffic as a result of needed broadcasts.

Client-server model

This model is presented in Figure 5.1. In this model there is a difference between the client that does the invocation of the server and the server that executes the job and returns the results to the client. From the figure it can be observed that servers can also act as clients for other servers. The problem of client-server architecture is that it scales poorly.

Peer-to-peer model

In peer-to-peer architecture all the processes play similar roles, so the hierarchy is flat. There is no distinction between client and server processes so all the processes

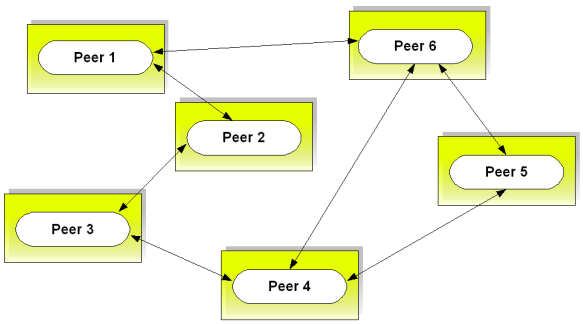


Figure 5.2 Peer-to-peer model

are peer processes. Peer-to-peer architecture is presented in Figure 5.2. The pattern of communication depends only on application needs.

Hybrid models

Aside from client-server model and peer-to-peer model there are also some variations. The main include:

- Multiple server model - since it is important for this thesis it will be covered in a separate section.
- Mobile code and mobile agents - In the case of mobile code, the code is downloaded to the client and from there the processing is done. In the case of mobile agent, a running program goes from one computer to another to fulfil a task.
- Network computer - it downloads the operating system and application from the remote server. The applications are run locally but the data are received remotely.

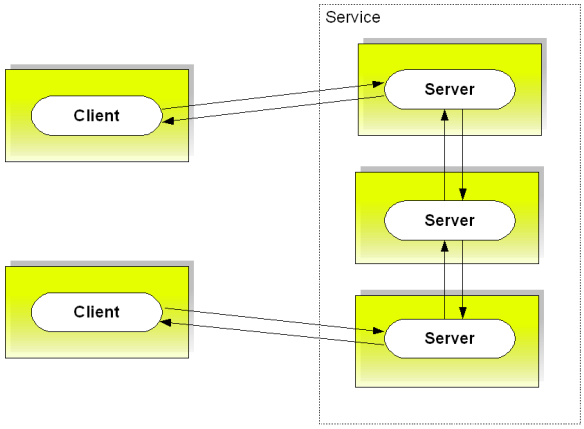


Figure 5.3 Multiple server model

Multiple servers model

Multiple server model assumes that there are multiple servers that are equal and that cooperate in order to provide service to a client. This architecture is presented in Figure 5.3. If the servers are tightly connected this architecture is also known as a cluster architecture.

This is a hybrid between client-server and peer-to-peer model since the client can be identified, but among the servers there is no strict hierarchy, so servers can collaborate together in a peer-to-peer fashion.

5.1.3 Fundamental models

All of the architectural models have some fundamental properties like interaction or security, so in order to describe these properties fundamental models are used. Fundamental models include the interaction model, the failure model and the security model. In the interaction model the communication and coordination between different parts of distributed system in time are described. The failure model models the types of failures that can occur with regard to their position in the communication

channel. The security model models the solution to the threats that are anticipated. All of these models are described in more detail in the following subsections.

Interaction model

In the definition of distributed systems it was said that the only communication between different parts is done by message passing. The communication channel has the properties of latency (time from the start of transmission to the start of the reception), bandwidth (total amount of data that can go through the channel in a given time) and jitter (variation of time required to deliver a series of messages).

Since the delay in the delivery of messages is not constant it is impossible to use reception times to calculate the exact ordering of the messages. This situation is presented in Figure 5.4. In order to have a logical sequence of events it is necessary to introduce a logical notation of time. In this notation the only thing that can be claimed is what event happened prior to another event.

Because of no absolute time there is a problem of synchronization. That means that in distributed systems there can be no discussion of perfect synchronization. But it is still possible to talk about synchronous systems, if the time boundaries for the processing, communication channel and the underlying machines can be well defined. The modeling of the system as a synchronous system is helpful, because based on the boundaries values timeouts can be defined. In asynchronous systems there are no time bounds, and solving of the design problems is harder, so most of the existing distributed systems are modeled as synchronous.

Failure model

The failure model defines the possibilities, reasons and consequences of failures. The taxonomy of failures used was introduced by Hadzilacos and Toueg [14], classi-

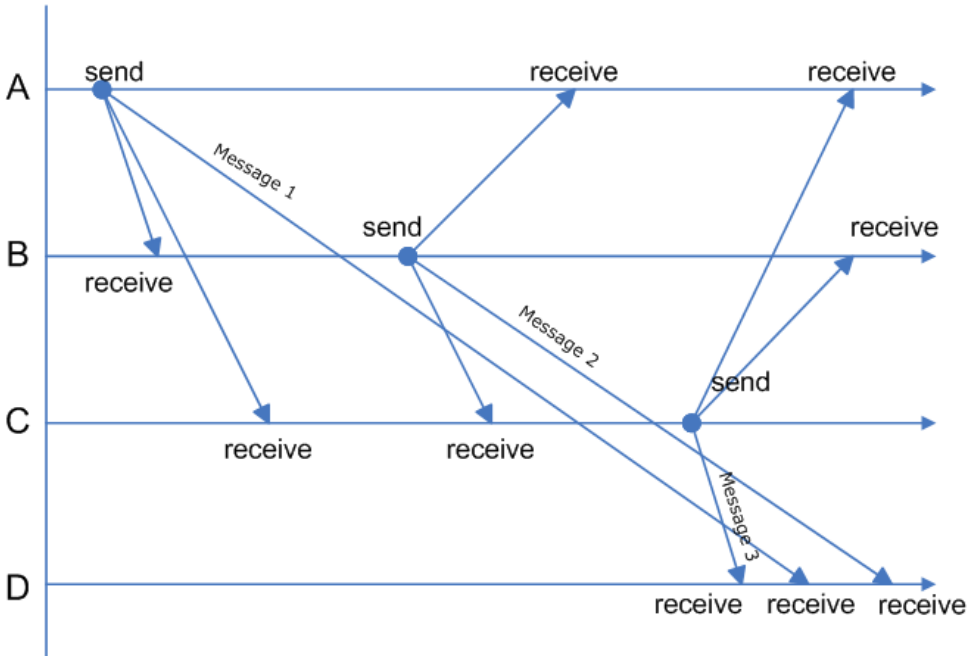


Figure 5.4 Reception times of messages in distributed system

fying the failures as omission failures, arbitrary failures and timing failures.

Omission failures are the failures in the communication channel. Based on the location of the failure in the communication channel, omission failures are classified as presented in Table 5.1.

Class of failure	Affects	Description
Send-omission	Process	Error occurred between process and outgoing message buffer
Omission	Channel	Error occurred between outgoing and incoming message buffer
Receive-omission	Process	Error occurred between incoming message buffer and process

Table 5.1 Omission failures

Arbitrary or Byzantine failures are the widest range or failures, introducing the failures that occurred without knowing why, or how. The error can be observed but the conditions that lead to the failure can not be repeated. Arbitrary failures in the communication channel include reception of duplicate messages.

Timing failures include the failures that occur because of a false notion of time. If the system is modeled to be synchronous and a timeout has occurred as a result of slow connection, this might be interpreted as a sign of failure.

Security model

When talking about distributed system security the topic is the security of the processes and communication channels. The securing of the processes starts with the definition of access rights. The processes of authentication and authorization are used

to enforce the access rights. The client request is checked before execution to ensure that the client has sufficient rights for requested service.

Communication channels can be secured using cryptography and authentication. The channels that have a layer of cryptography and authentication on top of the existing services are known as secure channels. Examples of secure channels are SSL and TSL. Except from authentication, secure channels are resilient to tampering, ensuring the message privacy and integrity. Most of the security threats can be eliminated using secure channels, but some still persist, like denial of service attack.

Designing a security model is not straightforward, since the encryption and authentication operations are expensive in terms of performance. A detailed analysis of threats, risks and consequences of the threats can be made using a threat model. Based on this model, the right security mechanism is used.

5.1.4 Middleware

Middleware is a software layer used in distributed systems that masks the underlying differences between system parts making them transparent to the upper layer programmers. The abstraction level implemented in middleware allows the programmer to interact with the distributed system using tools like remote procedure call or remote event notification. These tools simplify management of the distributed system providing a uniform view of a heterogeneous system. The position of middleware in the context of layers is presented in Figure 5.5.

Remote procedure call (RPC) is one of the oldest distributed system concepts. RPC function is to provide a transparency for the system by masking the fact that procedures called are implemented remotely. On the side of the client (calls the remote procedure) there must be a part implemented to recognize that the procedure is implemented remotely and on the part of the server there must be a way to get the

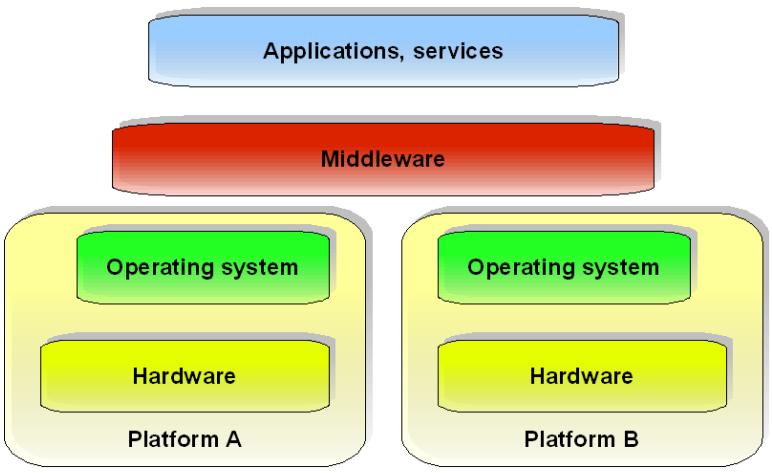


Figure 5.5 Middleware

arguments right from the client. By implementing this support for RPC, different programming languages on the client and the server can be used, as long as the conversion takes place when passing the arguments as well as results.

A specific type of middleware created to transparently provide access to shared resources on a very large scale is called grid. A grid is a collection of loosely coupled machines that cooperate together without fully trusting each other.

5.2 State of the art in distributed network simulation

Network simulation is a broad term and with multiple definitions. The term of network simulation is used in this thesis as a system that is capable of representing real world communication networks in terms of connectivity (topology) as well as communication (networking protocols and/or traffic).

A topology is a network model that consists of nodes and the description of how the nodes are connected. Depending on the type of network simulator there can be only one type of nodes or several types of nodes, specialized for certain purposes. The

connectivity between nodes is commonly described with links that connect one pair of nodes, which also have some specific properties.

Most network simulators offer specific network protocols as well as complete protocol suits to support the traffic generation and capturing. For instance, the TCP/IP protocol suite can be found in most of the existing network simulators since it is the most widespread protocol suit in use today.

Further on, there are two kinds of network simulators, off-line simulators, or just simulators, and emulators. The key difference is in the notion of time. Emulators use real time for processing while simulators use logical time.

Simulators use time notation to provide a correct sequence of events instead of using absolute or wall-clock time. The simulation executed in off-line simulators can be either faster or slower than in real-time systems. Network simulators such as ns-2 [7], GloMoSim [3] and PADS [20] belong to this category.

Emulators process events while keeping the notion of real time and are also called real-time simulators. They are designed to highly resemble real systems, so they need more resources than off-line simulators. Usually they provide connectivity with real systems and can use unchanged applications in the simulation. This category comprises emulators such as EMPOWER [26], Emulab [15], ModelNet [51] and Planetlab [5].

When comparing network simulators with so called testbeds, i.e. real networks used solely for experimentation in closed conditions, the benefits from the simulation are prevailing. These benefits are time and cost effective because they include easier experiment manipulation and data acquisition as well as less space and less equipment required.

As the network architecture evolves with time, requirements are made of the network simulators to be more and more scalable, so even large networks can be simulated [13]. With the emergence of new kind of distributed applications like grid and

peer-to-peer ([41], [38]) systems, the demands on the network simulators have been increased. Although small and medium scale network simulation provides valuable insight of the network behavior, this is not enough to draw the conclusion about the realistic behavior of the proposed new protocols, applications, mechanism or service. The scalability of the network simulators presents the limit in the scale of the network that can be successfully simulated.

To increase the scalability, simulators implement one of the following three methodologies:

- Simulation technology - by implementing more efficient programming structures, the scalability can increase, but using this approach the scalability can be increased only to a certain maximum value.

- Simulation model abstraction - by simplifying the model, the resulting demands will decrease, so in the end the realism of the simulation is being reduced.

- Computational power - additional computational or processing power can be added to the system and in that way the scalability is increased. The addition of the computational power can be achieved either by using a mainframe or by designing a distributed network emulator.

Although all of the approaches increase the scalability of the simulation, only the distribution of the network simulator offers a viable solution for a large-scale network simulation.

Simulators like ns-2 [27] use all of the stated techniques to achieve the higher scalability. Emulators tend to avoid the method of simulation model abstraction since the primary purpose of emulation is in offering realistic results based on the usage of a real code. But there are some exceptions to this rule (ModelNet [51]).

Some simulators started as single machine systems like ns-2 [27], but in time they

developed the distributed version. Others, like planetlab [5] and EMPOWER [26], are envisioned as distributed from the beginning and a non distributed version (not possible in Planetlab [5]) scales poorly.

There is also a class of network simulators that just combine more than one of the existing network simulators and offer a uniform interface to the user PlanetSim [29] and Emulab [15].

Along with the development of the general purpose network simulators that provide a support for large-scale application there is also a specialized class of purpose specific network simulators like MaSSF [21], designed for grid applications.

In the past few years, the accent in the distributed system design has changed from client-server models toward the peer-to-peer models. The reason for this behavior is found in the better robustness and scalability of peer-to-peer systems. These kind of systems are easily extendable mainly because of the lack of hierarchical approach and the absence of central authority.

Since decentralized architectures are more recent and less controllable than centralized architectures, the architecture of most distributed network simulators is based on centralized concepts (Planetlab [5], ModelNet [51] and Emulab [15]). In this approach client and server have well defined roles, and the management of the whole system is done in a centralized fashion. This kind of approach has a bottleneck on the machine that does the scheduling of the tasks.

5.3 Centralized vs. Decentralized

When developing a new distributed system of any kind, the first question is how the system will be managed. There are two main streams, going centralized or going decentralized. By analyzing the existing network simulators, based on the architecture used, simulators can be placed into a centralized or decentralized category. However,

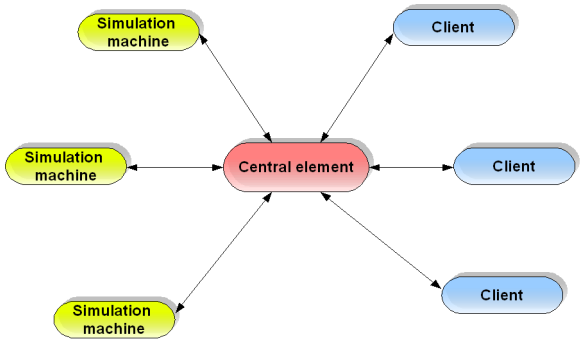


Figure 5.6 Centralized distributed network simulator

some of the systems like ModelNet [51] or PADS [20] require a manual setup of the distribution model and do not fall in any of the following categories.

5.3.1 Centralized architecture in network simulators

Centralized architecture has a central element that presents the bottleneck of the system. However, centralized architectures are more widely used since they offer easier management. The synchronization between elements is straight-forward, since there is one arbiter that has the view of the whole system and he makes all the decisions about the system. Centralized architecture is presented in Figure 5.6.

Netbed [47] is an integrated environment for simulation and emulation. Real network elements are intermingled with the simulated ones, each in charge of a different network portion. In the core of Netbed is a cluster system allowing time-shared and space-shared experimentations. Netbed architecture [47] contains a central element called masterhost that takes care of the web interface as well as database and SNMP management. Netbed evolved into Emulab [15], still leaving the centralized parts of architecture.

NTCUns simulator [43] is capable of simulating wired and wireless networks. It

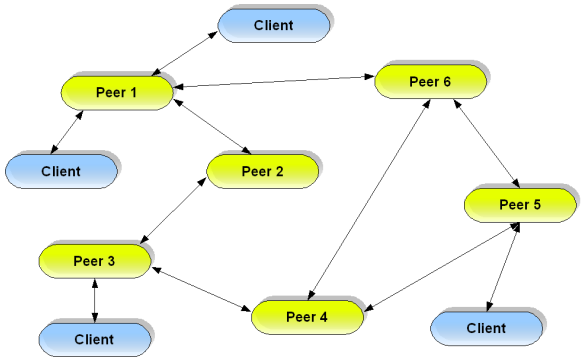


Figure 5.7 Peer-to-peer server architecture for network simulator

has an open architecture and is easily extendable. The architecture of NTCUns has a dispatcher as a centralized element. This dispatcher is the critical element of the system, and must remain alive at all times to manage the running simulations. All the communication between users and simulation machines goes through the dispatcher.

5.3.2 Decentralized architectures in network simulation

Most of today's attention in the design on distributed systems is focused in the field of decentralized architectures. The scalability problem with centralized architectures puts the limit on the size of the system that can still function with benefits. In centralized architectures the central element always presents the bottleneck of the system. Decentralized systems try to work a way around this limit and have a scalability that largely extends the scalability of centralized architectures.

Peer-to-peer architecture presented in Figure 5.7 shows one of the possible decentralized architectures. There is no central element, all of the machines are equal among themselves and there are ways of adding new peers or removing the existent ones without the need for a global lock of the system. Systems like chord [41] or pastry [38] present some of the implementations of peer-to-peer architectures. In them

there are developed algorithms that manage the tasks of adding new and locating the existing information.

Decentralized systems offer a lot of benefits mainly in the area of building scalable and cost effective systems. However, they are also much more complicated to control and in practice they are harder to design. In our distributed network simulator prototype decentralized architecture was implemented, since in the long run decentralized solutions tend to be self-organizing, minimizing the management overhead.

Chapter 6

Distributed network simulator based on IMUNES

In this chapter our attention is focused on our system, covering some of the design principles and problems encountered in the development of the system. Here the communication model of the resulting system is presented in more detail. This chapter can be of help to people dealing with the same or similar problems. Here the reader can find explanations why some techniques were used and others disregarded.

6.1 Our system architecture

The management problems that were identified through many different network simulators are:

- Resource management
- User management
- Experiment management
- Traffic management

Resource management provides for scalable and transparent usage of resources. Scalability means that new machines are easily added to the system and that addition of new machines will result in more resources available. Transparent usage of resources means that all of the resources of the machines in the system will be available in a uniform way. For meeting the scalability machines form a decentralized architecture. The architecture used takes care of addition of new machines that present new resources to the distributed system, as well as removal of machines from the system and failure management if one of the machines fails. To achieve transparency the

naming scheme used for addressing the resources, i.e. machines, is independent of their physical location. In our distributed system available machines are defined in two different ways: explicitly and implicitly. Explicit means that there is a closed set of well defined machines that is given during the configuration. In this case there is no addition of new machines in the system. Implicit means that each new machine searches the given IP address range in pursuit of an already existing set of machines and joins the set if one is found, otherwise it waits. In the case of crash or sudden leave of the machine, all the experiments that were partially or fully conducted on that machine are stopped.

User management takes care of all the users that are available. This includes the tasks of authorization, authentication and propagation of rights of existing users. The authentication mechanism can be login and password authentication or a key authentication. Propagation of rights occurs when a user asks the system for more resources. If there is a machine willing to accept this user, the user rights should be propagated to that machine so the user can gain access to more resources. Furthermore, the user management should also takes care of new users, using the same mechanism to gather the resources for a new user. After a user account is closed, another notification process takes place to remove all the user information and processes from the system. This event driven behavior of propagation of rights resembles the publish-subscribe systems. Authorization of a user implies that a user can change only the experiment that he owns, and that he cannot start an experiment on machines he does not have the sufficient rights for. In our prototype version users are defined statically, and there is no implementation of propagation of rights. Authorization is implemented on the level of experiment. Authentication of a user is currently implemented with a user providing only his username.

Experiment management presents another important issue, since tracking of the experiments and keeping the information about them should be available through

a simple, but efficient mechanism. Experiments need to have a unique identifier so that the information can be kept without interference between different experiments. Each experiment can belong to only one user. Creation of a new experiment is another event, and upon the creation of a new experiment all of the machines that provide access to the owner of the experiment should be notified. This information about new experiment is kept on all machines available to the user who created this experiment. New experiment name is bound to the machine that user is communicating with instead of user that is creating the experiment; this is done in order to circumvent the racing conditions in case of a simultaneous creation of more than one experiment by the same user. Stopping the experiment removes all the information about experiment form all the machines, i.e. the global state of the experiment is cleared.

In traffic management it is important to separate each traffic flow, one from another. This is required since there might be more than one experiment using the same addresses. Our system is modeled to prevent that kind of situations. For traffic management traffic separators must be used. Currently, there are two options for traffic separation: UDP tunneling and VLAN tags. Usage of tunneling introduces additional delay in the system. However, at the same time it separates the simulated/emulated traffic from the real Internet traffic and separates one traffic flow from another. For creating a UDP tunnel two machines need to come to an agreement. Usage of VLAN tags reduces the scope of the distributed network simulation to a local area network. But there is lower overhead and less chance of IP fragmentation. Another downside of using VLAN tags is that all the machines in the set should reach an agreement which VLAN tag to use.

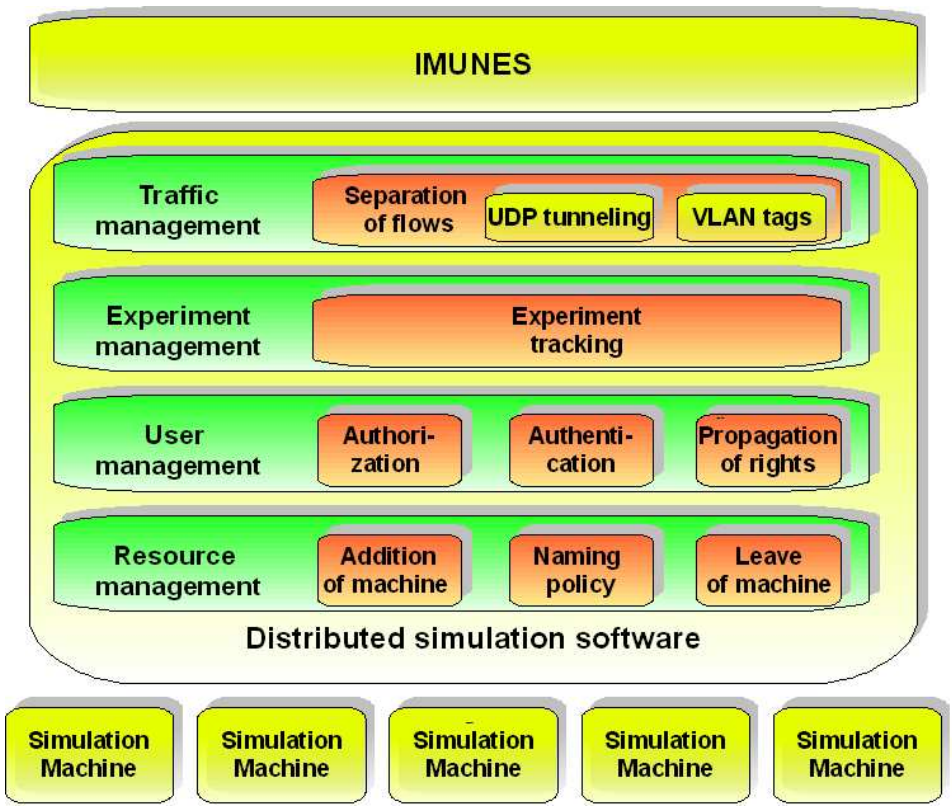


Figure 6.1 Architecture of our distributed network simulator prototype

6.2 Naming conventions

In our distributed system it is possible to identify certain objects. These objects are servers, users, experiments, nodes and links. For a system to be scalable, naming conventions must be applied so objects can be easily accessed and created.

6.2.1 Servers

All the servers have the information about all the other servers and all the identifiers for servers start with the letter 'h' followed by the identifier. The identifier is usually an IP address, but it can be anything else.

In the case of open architecture the system is deployed on top of a local area network. A new system is created openly with all the machines in the local area network broadcasting the request. Each new machine is added to the set of servers without additional security checks. For this architecture a unique identifer is based on the machine's IP address to avoid global locks, but we keep additional information about the IP address on each machine.

In closed architecture servers are defined statically and are assigned unique identifiers in the configuration file. If there is a request from a machine with an IP address not defined as a part of the allowed servers list, the request is not granted. Upon the starting of the server, the software instantly tries to connect to all the machines specified.

6.2.2 Experiments

Unique names for servers are important for the creation of new experiments and access to the old ones. When a new experiment is created, the experiment's configuration file is transferred to the server. The server assigns a unique identifier to that experiment and sends this information back to the client. So, each time the client

accesses the experiment he does that through the experiment identifier.

The name space of the experiments is divided among all the servers. Each server has a limited number of experiment identifiers, so there is a limited number of experiments that it can start. Each experiment identifier in its name contains also the server identifier of the server where the experiment was created. So, there cannot be more than one experiment with the same unique identifier.

6.2.3 Users

All the users in the system need to have a unique name. At the moment, all the users are defined statically in the configuration file. In the future, the user's name space can also be divided, or a global lock for defining new users can be used.

6.2.4 Nodes

Each node in the distributed system is uniquely identified with the node name and the experiment to which the node belongs. The node name must be unique within one experiment. Node name starts with the letter 'n' followed by the node number (for example $n12$). Resulting kernel structures (virtual images and netgraph nodes) have names that consist of an experiment identifier, a sign _ and the node name (for example $e0001\_n12$).

6.2.5 Links

Each link in the distributed system is uniquely identified with a link name and the experiment to which the link belongs. The link name is unique within one experiment. The link name starts with the letter 'l' followed by the link number (for example $l12$). Resulting netgraph nodes, ng_pipes are named starting with the experiment id, followed by the sign _ and first node identifier, sign - and second node identifier (for

example $e0001\_n12 - n14$).

6.3 Remote procedure call

For communication between client and server as well as for inter-server communication the Remote Procedure Call was used. Remote Procedure Call is the earliest and best known model that allows clients to call procedures on the server programs that reside on separate machines [14]. Since no standard module for RPC for Tcl/Tk exist, we wrote our own implementation. In this implementation for each new server there are three procedures: connect procedure, evaluate procedure and disconnect procedure. Connect procedure is called when the communication is established. Evaluate procedure is called before the execution of local procedure called from remote machine. Disconnect procedure is called when the connection is broken, either deliberately or due to failure.

There are three ways of calling remote procedure using RPC: synchronous, asynchronous and with callback. Synchronous calls are non-blocking calls, which bring the result when it becomes available. Asynchronous calls return a variable, where the result will be stored as soon as it becomes available. Callback calls define a local procedure that will be called with the results as an argument.

6.4 Client-server communication

In client-server communication the client sends the requests and the server responds. The client sends some of the predefined messages, while the server keeps the state of the communication. In this subsection the messages interchanged between client and server will be described, as well as the state machine of the client-server communication. In the end security issues are discussed.

6.4.1 State machine of the client-server communication

Client-server communication is presented in Figure 6.2. This figure shows the states in the communication and what messages are possible in each state.

States and messages

The possible states for the server are: Init, Normal, Topology_get, SPL_get, Topology_rcvd and Execute. In each of these states there is a list of possible messages. By traversing the states the server executes the requested actions that it has sufficient rights to do. All these states will be described, as well as possible messages for each of them.

The Init state is the initial state of the server. The only message that is acceptable in this state is user authentication message. After the user is recognized as an authorized user of the cluster, the server goes to the state Normal.

Normal state is the state from which the user can start two different actions. He can try to load the experiment that is already deployed on the distributed system, or he can start a new experiment. When a user wants to load an existing experiment, he can ask the system to give him the list of all his active experiments, so he can choose which one he wants to load. The list of all experiments is obtained with the getExpList message, and an experiment is loaded with the eLoad message. From the state Normal, if the experiment is going to be distributed among different machines, the user can ask for a list of all servers available with the message getServerList, to do the mapping between parts of the experiment and available machines. With the createConfFile message the user initiates the process of starting a new experiment.

In Topology_get state, all the received data is part of the experiment topology and it is automatically put into a file. Only the message createSPL will be recognized as not being a part of the topology, but instead as start of the mapping. From

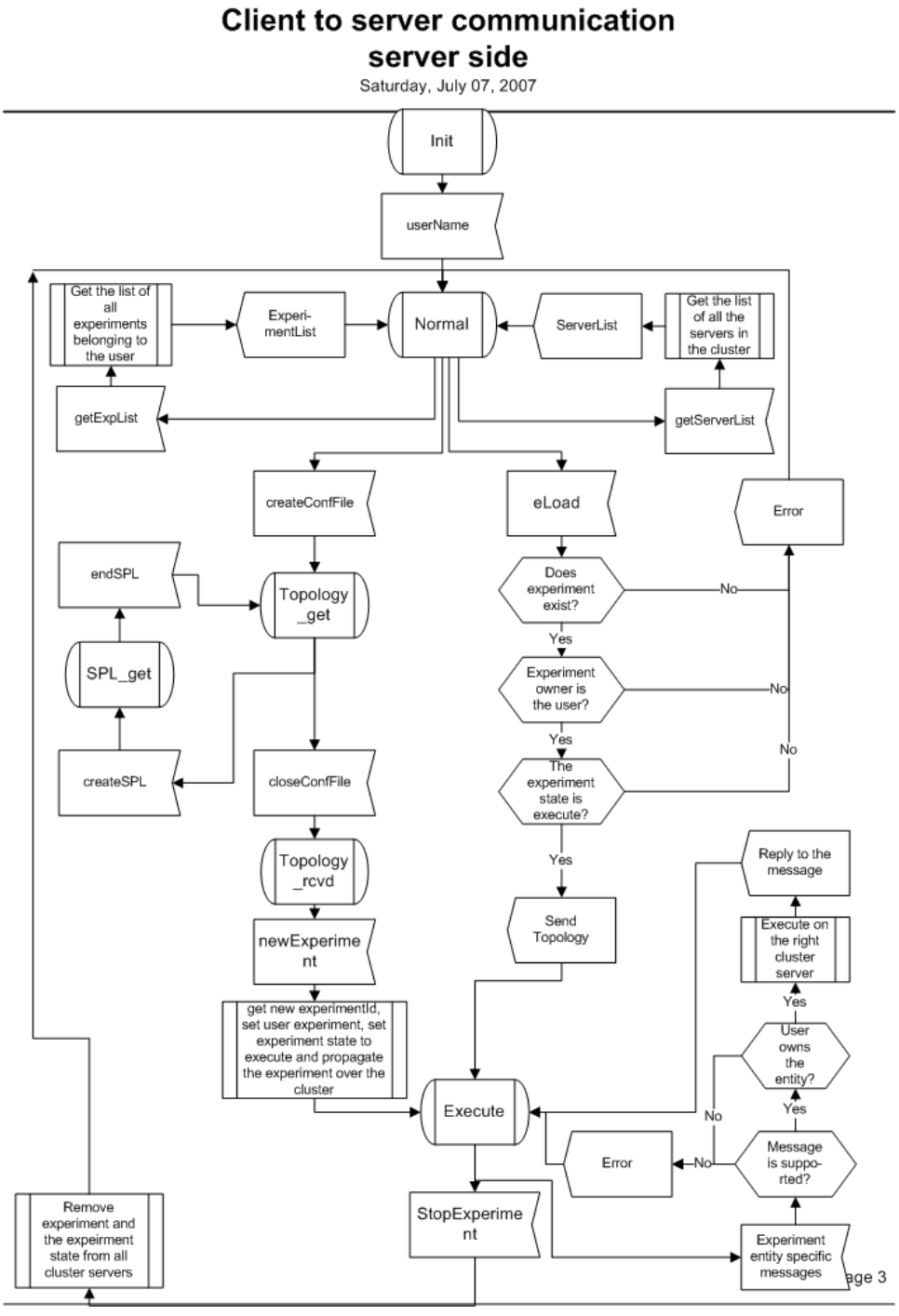


Figure 6.2 Client-server communication

Topology_get state with the message closeConfigFile the server goes to the state Topology_rcvd.

SPL_get state can be reached during the transmission of new experiment topology. In this state the mapping between different parts of the experiment and the available machines is received. This information is crucial for the experiments that want to use more than one server. After this information is transmitted, the server changes the state back to Topology_get state with the message endSPL.

Topology_rcvd state is an intermediate state in which the server finds itself after successful reception of new experiment topology. The only message allowed in this state is newExperiment, which causes the experiment to be started.

After the experiment is started, the server goes to the Execute state. This state is at the same time assigned to the new experiment as well. When the server is in the Execute state, it can receive different sets of specialized messages (listed in Table 6.1). From Execute state with the message stopExperiment the server returns to Normal state.

6.4.2 Security issues

As a transport layer security mechanism there is a support for optional SSH tunneling, providing that the information sent between client and server remains confidential. This way all the communication goes through a secure channel, having both cryptography and authentication support.

On the application layer our application is secured by additional checks. For each received message there is a check if the message is supported and if the user has sufficient rights for requested action.

Message	Meaning
vimageShellServer	Searches the servers and returns the IP address of the server that simulates the node for which a user wants to open a shell
setNodeDir	Removes and recreates a node's temporary directory
setNodeMTU	Sets the nodes MTU value
nodeBoot	Starts the node booting
stopNodeProc	Stops all the processes on the node
stopNodeIfcIPv4	Removes the IPv4 address from the node's interface
stopNodeIfcIPv6	Removes the IPv6 address from the node's interface
setLinkIfcQDisc	Sets the queueing discipline on the interface
setLinkIfcQDrop	Sets the queue dropping policy on the interface
setLinkIfcQLen	Sets the queue length on the interface
setLinkParams	Sets the parameters of the link
findShell	Returns the list of available shells on the machine where the node is simulated

Table 6.1 Set of messages

6.5 Inter-server communication

Servers communicate with each other through a set of messages. All communication goes through RPC, and before each command is executed, the global function is called to check the received messages.

6.5.1 Global state

Global state is held on each machine in the server. The Global state consists of the list of all users, a list of all the servers and all the experiments. For each server the information about the server IP address, server name and interface that is used is kept, as well as the list of sockets that are used for communication with that server. The user information is the information about the experiments that the user has. Experiment information is the information about the topology, the list of all the servers and parts of the topology that will be simulated. On each server as a part of a global state there is also information about all the nodes and links of all the experiments.

Global state changes after the addition of a new server, with server crash, with creation of new experiments or by stopping the experiment.

6.5.2 Communication model

Servers communicate with each other by using the RPC model. For each connection there are three different stages: connection establishment, command and connection lost. These events are supported in our RPC implementation as different procedures.

The connection phase starts as soon as the server application is started. The started server gets the network that it scans in lookup for other hosts. When it finds the other hosts it tries to connect to them. If the connection is successful, it adds

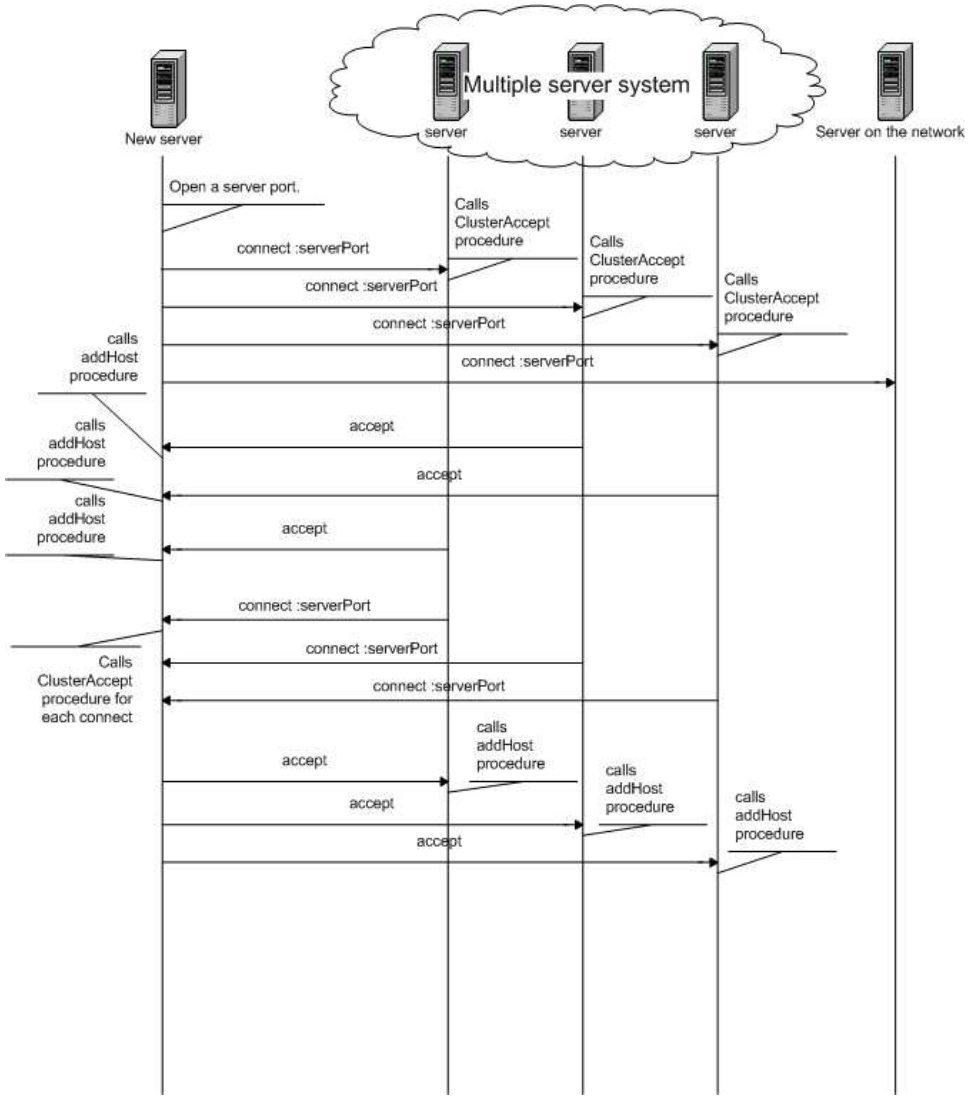


Figure 6.3 Connection establishment phase

a new host to its hosts_list, and keeps the socket information. On the other hand, if the server is already working, and a new host initiates the connection, the server accepts the connection, and if the new host is not on the list he adds it to the list, and initiates a new connection to that server. The connection establishment phase is presented in Figure 6.3.

The command phase is the normal functioning of the server. The server in the command phase can act in two different ways. One way is when a server communicates with the other server on its behalf, another way is when the server acts on behalf of the client, asking for specific commands to be executed. When a server communicates on its own behalf, it sends specific messages to the other server concerning the experiment as an undivided entity. In this form of communication the server updates the global state of the cluster, and the same messages are relayed to all the machines in the cluster. One example of the server-server communication is presented in Figure 6.4. These kind of messages are shown in red. When the server issues specific commands to just one server, the server is just relaying the command from client to the server emulating some part of the experiment. This situation is presented in Figure 6.5.

The connection lost phase is defined as the closed state of the socket that is connecting the machines, and is reported by RPC. When a server finds out that one of the server machines is no longer available, the machine closes the connection and immediately stops all the experiments that were run on the machine that is no longer available.

6.5.3 Critical resources

The critical resource in our case is the physical link. Since it must be possible to simulate more than one link on one physical link, there must be a way of separating the simulated link traffic instances one from another. To be able to do that, two

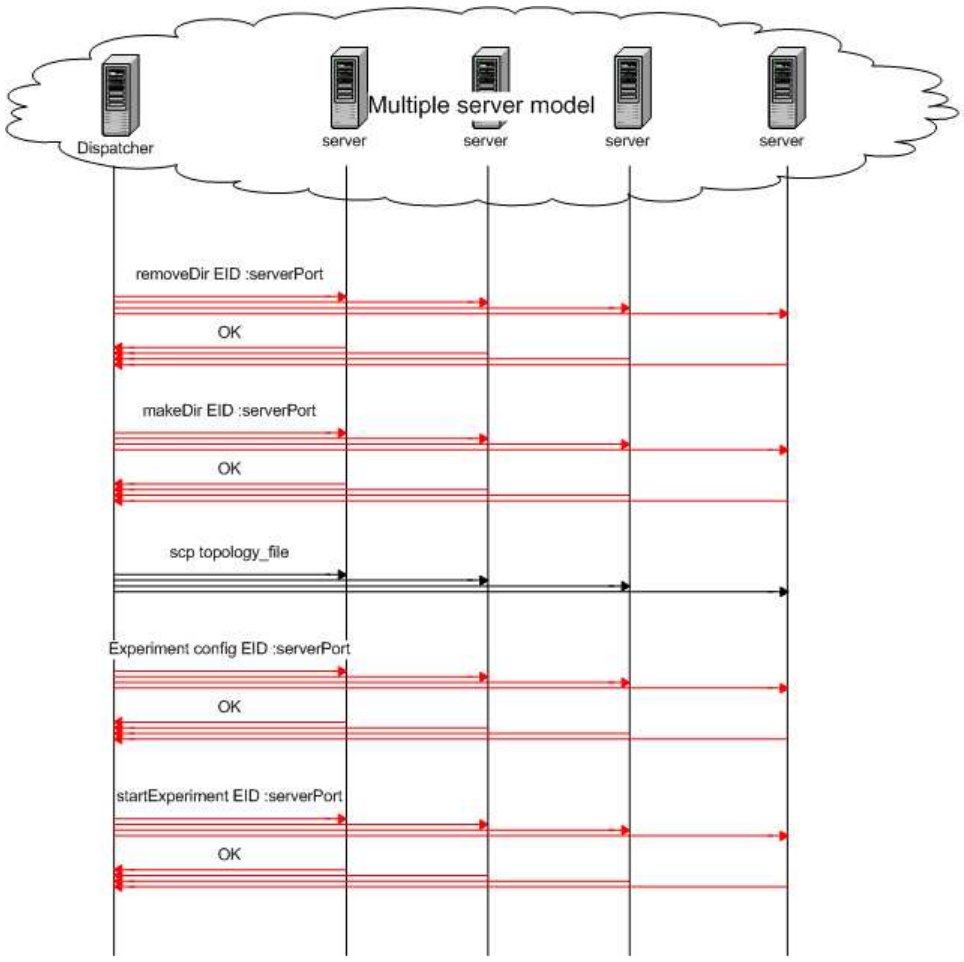


Figure 6.4 Creation of new experiment

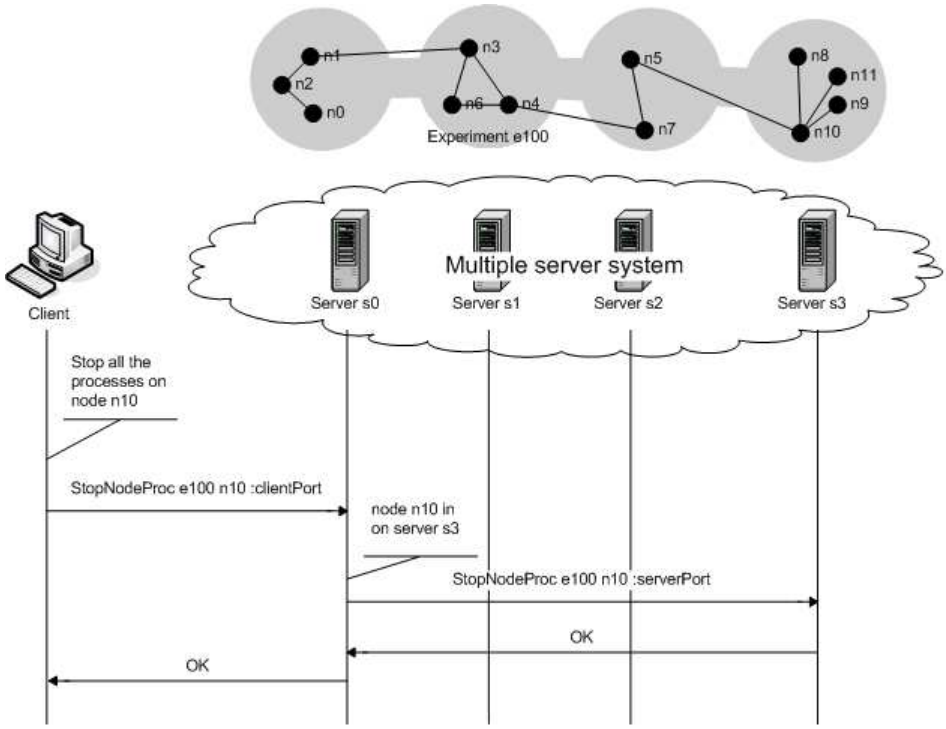


Figure 6.5 Stopping the node's processes

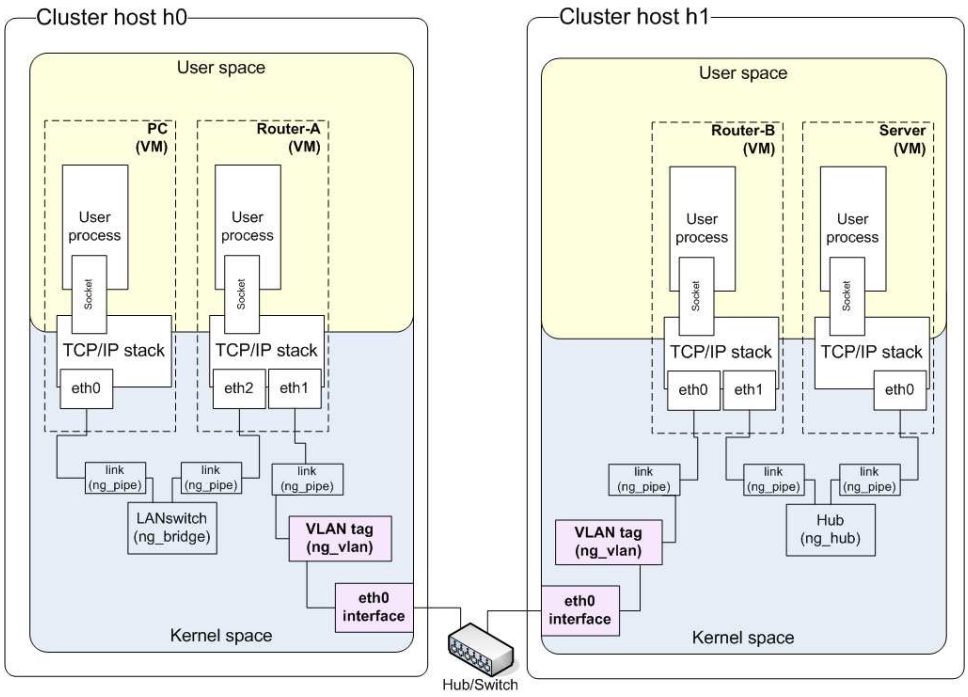


Figure 6.6 Distributed experiment with VLAN

different techniques are used, one of which is separating simulated link traffic with VLAN identifiers and the other is using UDP tunneling.

When using VLAN identifiers there is a need for synchronization in order to be able to assign a new VLAN identifier that would be unique in the system. This synchronization is expensive and the usage of VLAN identifiers restricts the system to functioning only in local area networks. The advantage of this approach is that the packets do not get fragmented on the network. The view of the emulated system with VLAN identifiers is presented in Figure 6.6.

UDP tunneling does not require full synchronization, but it does require synchronization between two servers that emulate nodes on both ends of the link. Because the packet goes back to the TCP/IP stack, this solution is more expensive, not to

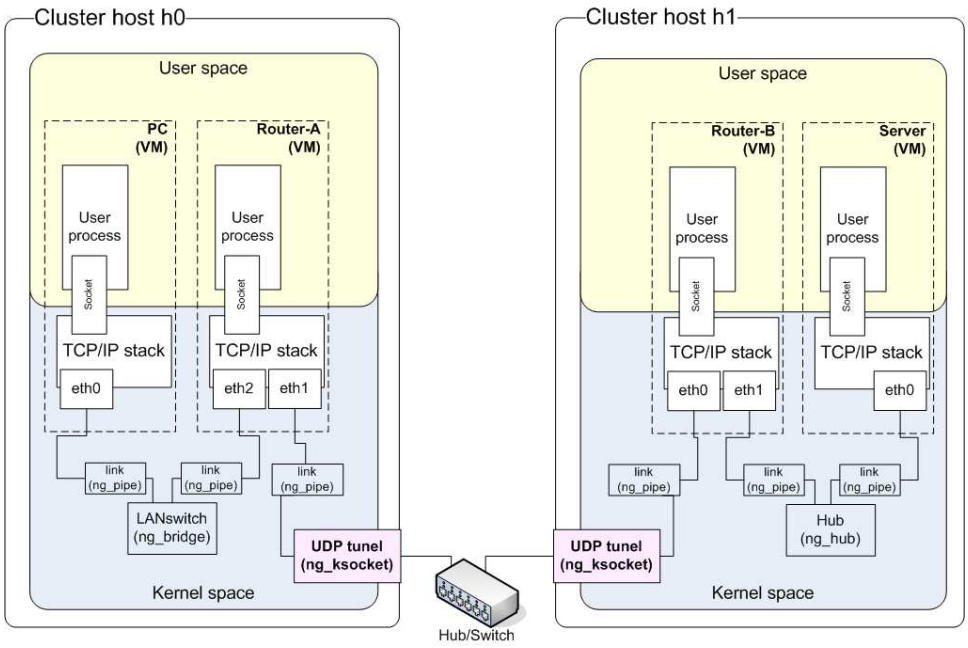


Figure 6.7 Distributed experiment with UDP tunnels

mention the fragmentation costs. But it provides good results as presented in Chapter 7. The view of the emulated system with UDP tunnels is presented in Figure 6.7.

6.5.4 Security issues

For transport layer security a secure tunnel is used. The implementation of secure channel that was used is a SSH port forwarding mechanism. This mechanism ensured that the other server is authenticated and that communication is encrypted.

For application layer security, a list of possible messages is used, so each message received from another server is checked before execution. As can be observed, servers do not fully trust each other.

Chapter 7

Results

7.1 Introduction

The performance of each distributed system is evaluated through standard measures. The objectives which a distributed system is trying to meet are described in Chapter 5. Evaluation metrics such as scalability, response time and emulated network performance were used to measure the performance of our distributed network emulator.

For measuring scalability and responsiveness of a system a setup with large experiments was required. To show the worst performance of the system, a network of routers was created, each running a dynamic routing protocol and introducing the highest memory and CPU load on the system. Created networks had up to 1000 routers and measured the performance in terms of memory usage, CPU usage, establishment time and tear down time. For comparison a network of up to 1000 nodes of type PC was created, since PCs are the lightest consumers working on a network layer.

The emulated network performance was measured in terms of network latency and throughput. Some of the simulated links in a distributed simulator go through real links and are separated one from another using two different technologies available in our system: vlan tags and UDP sockets. Both of these technologies have their benefits and drawbacks. Both methods of this methods were compared to the performance of standard IMUNES having as a reference data from the testbed. For latency and throughput measurement the setup was a simple experiment with only three nodes and the network performance was evaluated with standard networking applications.

The measure of scalability actually measures the effective usage of the resources.

As will be presented, our system is scalable, and this scalability is proportional to the number of nodes. The responsiveness time should be as low as possible, but there is an overhead brought by the network characteristics that does not depend on the design. For emulated network performance it is expected to observe that decrease in performance will be within reasonable limits, still offering a good match between an emulated system and a real one.

7.2 Scalability and responsiveness time

The scalability and responsiveness time of the system were measured. The scalability is an important measure for all distributed systems, since it tell us about the system's limit in size. The scalability of the system was measured by measuring the CPU and memory load of the dispatcher machine and the results obtained were analyzed. The responsiveness time is also an important metric of the system since it describes how easily the system can be manipulated. This is also the time the user should wait for the resources to be allocated. The scalability is described first, and later we move to responsiveness time.

For measuring the scalability a large network with several hundreds of nodes was designed. The example of the resulting topology is presented in Figure 7.1. The goal was to find out the load that the resulting system imposes on the dispatcher machine in terms of CPU load and memory usage. The parameters of the system are given in table 7.1.

In the experimental setup the distributed system was created with a given number of machines acting as servers and one remote machine acting as a client. First, the experiment topology was created and divided into partitions on the client machine. Second, the topology is transferred through the network to the dispatcher server. The dispatcher server dispatches the topology to all the other servers and starts the

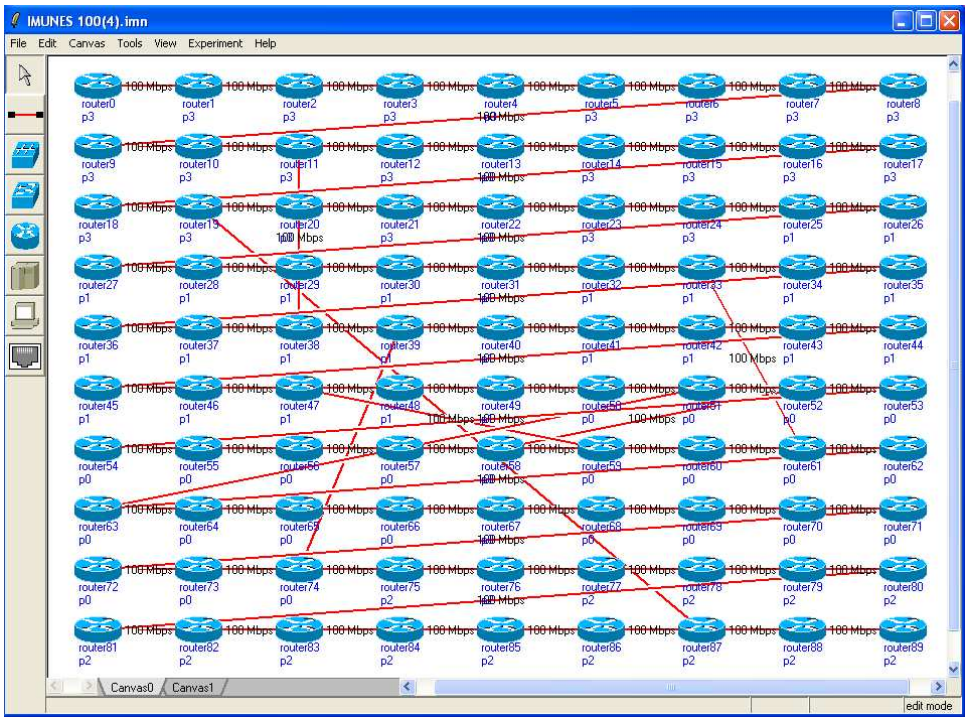


Figure 7.1 Example topology used for measurement of scalability and responsiveness time

Parameter	Value
CPU	Intel Celeron(R) 2.66GHz
RAM memory	1GB
Network card	100Mb/s
Operating system	4.11 FreeBSD, network boot
Number of machines	from 2 to 10

Table 7.1 The setup of the machines used for the experiment

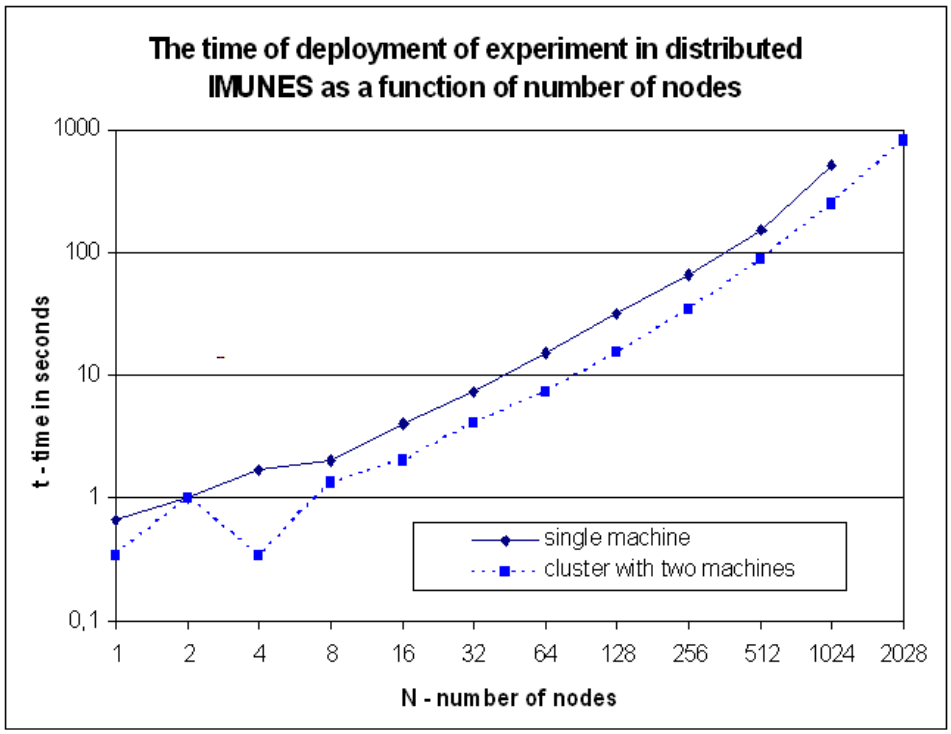


Figure 7.2 Experiment establishment time for prototype implementation

experiment. The communication pattern between the client and the dispatcher and from the dispatcher to all the other servers is given in Chapter 6.

The CPU load and memory load on the dispatcher machine were measured after several seconds, allowing the machine to come to a steady state.

The results of the prototype implementation of the distributed system are published in [33] and are presented in Figure 7.2. The topology created for the experiment published there was a simple chain topology with an equal number of each type of IMUNES nodes.

From Figure 7.3 it can be observed that the CPU load increases with the number of nodes in the experiment. The CPU load growth is not linear. With the growth of

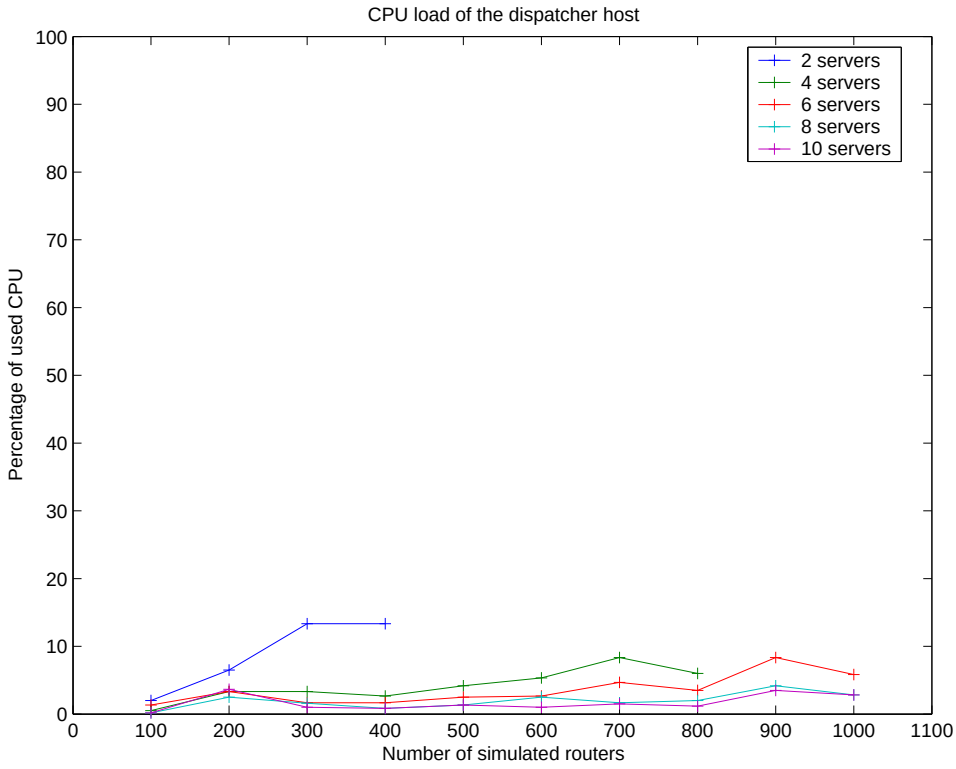


Figure 7.3 CPU load of the system

the network an increase in the data that has to be processed can also be observed. The increase in the data is not linear, so the resulting CPU load should not be linear neither. From Figure 7.3 it is obvious that CPU load does not present the bottleneck, since with the maximum load the CPU usage goes to less than 20 percent.

In addition to the CPU load the memory usage (Figure 7.4) was measured. The memory in use grows with the increase of the number of nodes in the experiment. The experiments were run on network booted machines, which use RAM memory for keeping the filesystem as well as for standard running data. There is also no swap memory since the machine's hard drive is not used at all. The memory constraints in that sense are very hard, and if a system tries to use more memory then currently

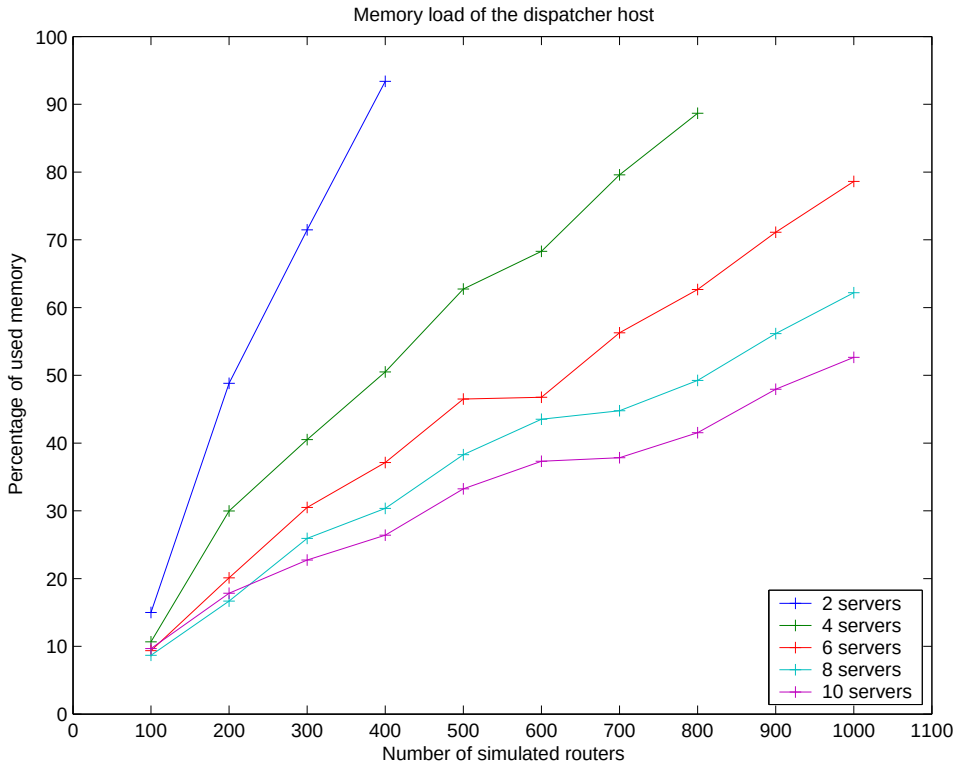


Figure 7.4 Memory load of the system

available, the machine will crash. By keeping the machine running for a longer period of time the memory also gets used because the machines were booted from the network, and in the consecutive runs the later ones will present more memory usage than previous ones.

From Figure 7.4 one can observe that memory usage grows with the number of nodes, and when the memory usage is correlated with the number of nodes, it can be observed that the memory grows proportionally to the size of the experiment. So, each new emulated node uses some predefined amount of memory. In table 7.2 we can find the parameter of proportionality representing the memory required by each node of the type router. We fitted the curve of memory usage with a linear function.

Number of servers	Slope	% of memory per router	total memory per router
2	0.25785	0.5157	5.157MB
4	0.10592	0.4237	4.237MB
6	0.07625	0.4575	4.575MB
8	0.05593	0.4474	4.474MB
10	0.04424	0.4424	4.424MB
Memory usage per router: (0.4573% $\pm$ 0.0349%) of 1GB			

Table 7.2 Router memory consumption

The results obtained with routers are the worst case scenario, since each router runs a dynamic routing protocol. Just for comparison, topologies with PC type nodes of the same scale were created. The results are in Figure 7.8. From that figure one can see that a network topology with PCs only, scales much better since the memory usage has been reduced substantially.

For measuring the responsiveness the same setup was used and the time required to establish the experiment was measured as well as the time to tear it down. The time in both cases is measured in seconds. In Figure 7.5 one can observe drop in establishment time when the simulation is distributed over a larger set of machines. The most significant drop is observed when increasing the number of servers from 2 to 4. In this case the time is reduced by a factor of 2. From that figure it can also be seen that there is a threshold value for which the distribution reduces the establishment time. This kind of behavior is visible from the data for 100 nodes distributed over 10 servers. In this case the establishment time increased with the number of machines, because the distribution load is bigger than the processing load.

Figure 7.6 shows the time required for creating the experiment. As observed from the figure, the time for the highest number of nodes and lowest number of machines is

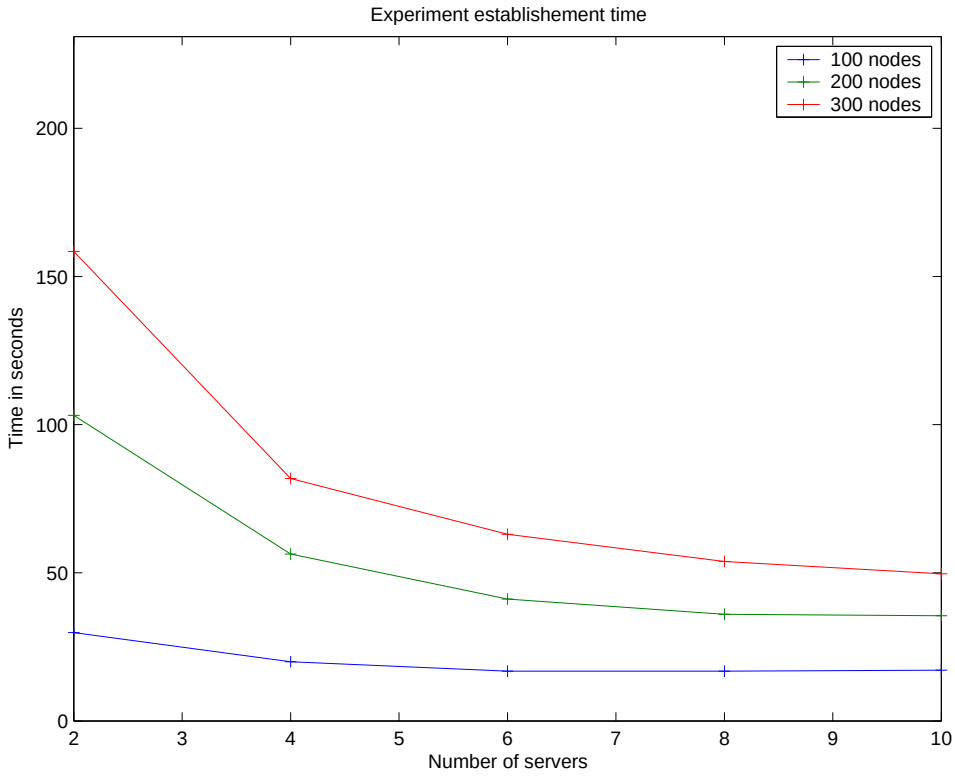


Figure 7.5 Establishment time

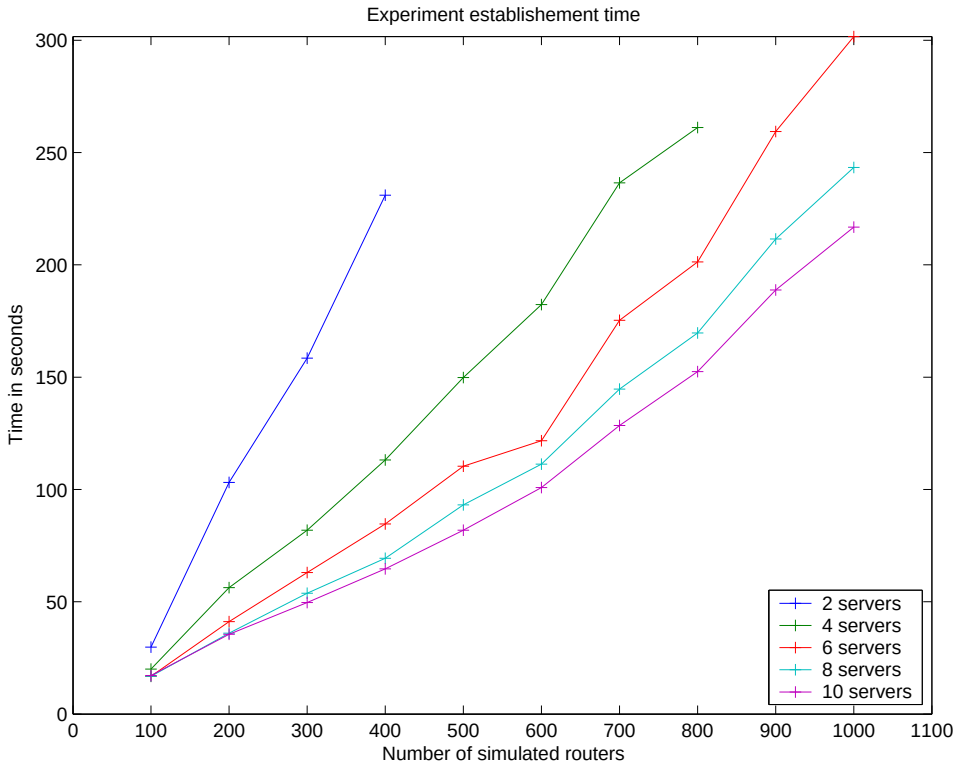


Figure 7.6 Experiment establishment time

around 5 minutes. This time is suitable for long term as well as short term simulations. Although in a distributed simulation the whole configuration file must be sent to all the machines, the overall time for distributed system employment decreases with the increase in the number of machines. It can also be observed that the increase in time required is not in a linear relationship with the number of simulated nodes.

Termination time for an experiment is presented in Figure 7.7 and we can see that the improvement in terms of time is neglectable. This is because the overhead of the distribution to more than one machine is of the same order as the decrease in the size of the network. From this figure one can observe that there is a linear relationship between the termination time and the number of nodes and the termination time is

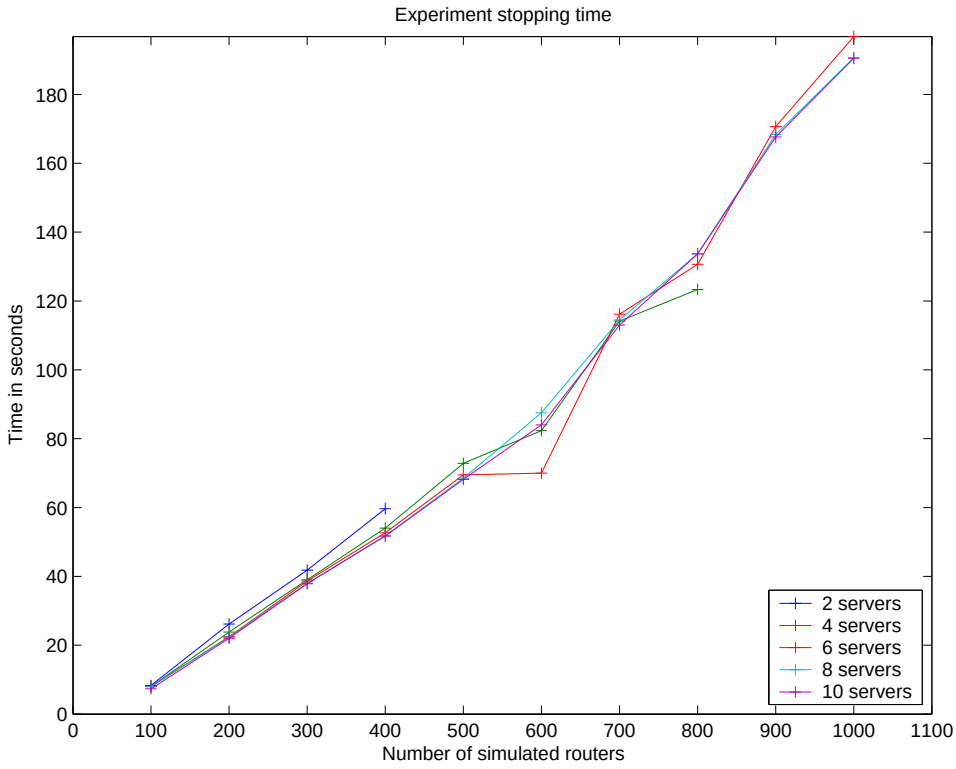


Figure 7.7 Experiment termination time

not influenced by the increase in the number of machines available.

Figure 7.8 shows the same results for a topology of the same scale, but with PCs instead of router nodes. As one can observe, the memory and CPU load is very low, keeping the system very scalable. Although this scenario gives much better results, it is not a real scenario, since there is no point in simulating a network in which there is no activity.

7.3 Performance measurement

The distributed simulation increases the resources available, but the price for this is lower controllability of the environment. The links simulated over real physical

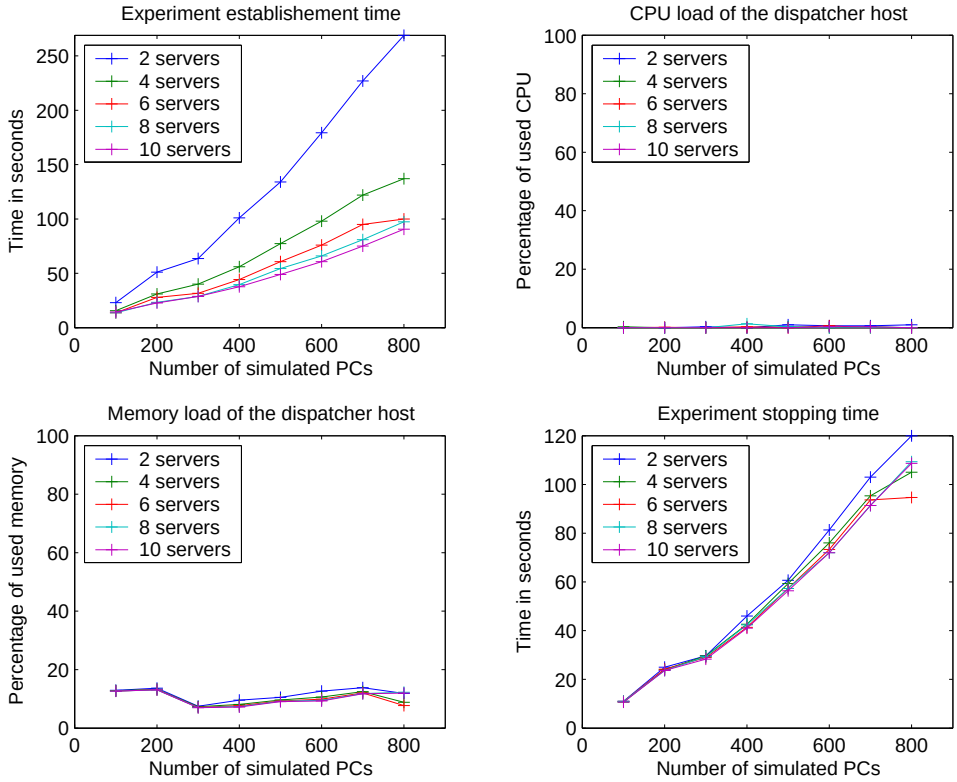


Figure 7.8 Scalability and responsiveness of PC network

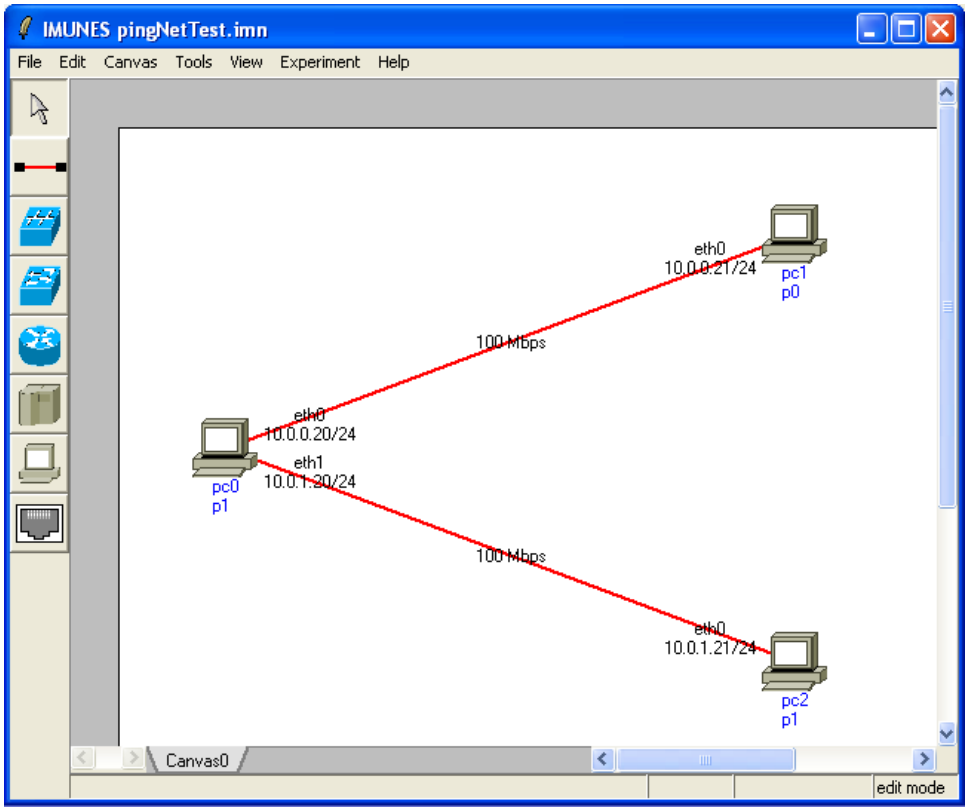


Figure 7.9 Simulation scenario

links have characteristics that differ from the stated values. In order to measure the impact a simulation scenario as presented in Figure 7.9 was created.

The performance measurements used are standard network performance measures, latency and throughput. These measures are textbook measures ([30] and [40]. For measuring latency round trip time (RTT) was used and for throughput the actual throughput of the link was used. The round trip time is the time required for a packet to leave one system, to be processed on another system and come back. This time consists of transmission time, propagation time and processing time. Throughput is a measure of how many packets (bits) can be received in a unit of time.

Four different scenarios measured were:

- Two nodes connected with a direct link and simulated on one machine. From Figure 7.9, these are nodes n0 and n2.
- Two nodes connected through a direct link, but simulated on two different machines. From the figure, these are nodes n0 and n1. The simulated link between two nodes is implemented using VLAN tags.
- Two nodes connected through a direct link, but simulated on two different machines. From the figure, these are nodes n0 and n1. The simulated link between two nodes is implemented with UDP sockets.
- Two machines connected through a LAN connection.

The parameters of the system are given in Table 7.1 and the results are given in Table 7.3. The measurements were repeated 5 times. For calculating round trip time measurements were obtained through ping program and for throughput netperf application was used.

The results indicate that VLAN implementation has better performance in terms of lower RTT as well as higher bandwidth. This is due to the fact that for VLAN implementation each packet is processed on link layer instead of going again through network layer. For VLAN tags there is a space envisioned in an ethernet frame, so one packet from the simulator is equal to one packet on the network as can be observed from Figure 7.10

That is not the case with UDP tunneling, where the resulting packet can exceed the MTU size, and get fragmented, as presented in Figure 7.11. When a UDP tunnel is used, the system is operating on a higher level allowing the packets to be transmitted to any destination machine as long as there is a route to that machine.

Environment	Mean value	Standard deviation	error
Round trip time (RTT) measured through ping (64B) in ms			
IMUNES	0.0266	0.0053	72%
Distr. IMUNES (vlan)	0.1374	0.0044	44%
Distr. IMUNES (UDP tunnel)	0.1538	0.0015	61%
Real system	0.0952	0.0008	0%
Throughput measured with netperf TCP stream test (10s) in Mbit/s			
IMUNES	94.1760	0.0055	0.45%
Distr. IMUNES (VLAN)	88.9380	0.9178	5.14%
Distr. IMUNES (UDP tunnel)	88.0840	0.1504	6.05%
Real system	93.7560	0.1060	0%

Table 7.3 Network performance measurement

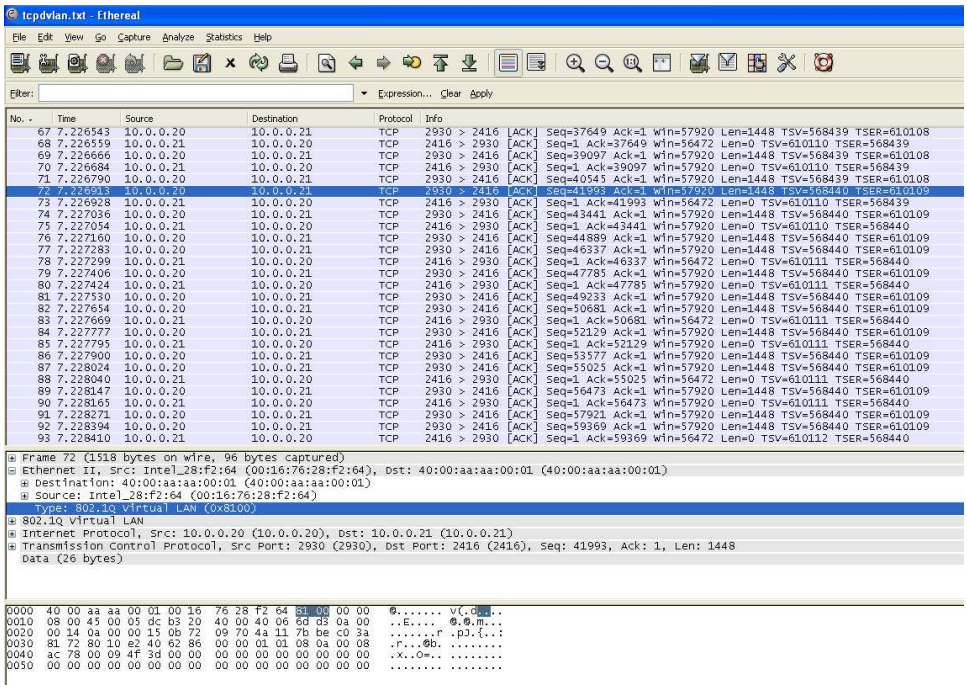


Figure 7.10 Packets on a real link going form one simulated node to another when the distribution is done with vlan tags.

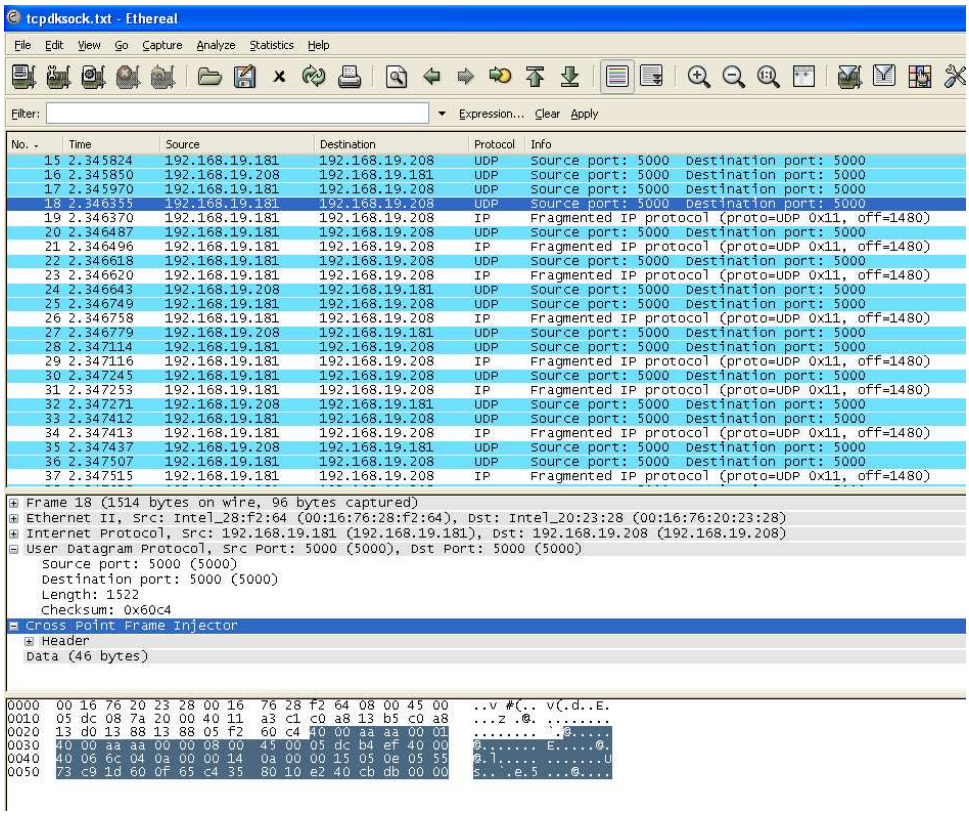


Figure 7.11 Simulation scenario

We can also observe that the RTT is approximately equal to the sum of RTT of a packet simulated on one machine and real RTT on the physical link between the machines. So, as expected that the processing time for VLAN is smaller than for UDP sockets, bringing a better performance for VLAN than for UDP tunnels, but UDP tunnels as well do not introduce a significant overhead. For both methods of sharing a physical link among simulated links, the processing time based on Formula 7.1 was calculated.

$$t_{proc} = t_{tot} - (t_{imun} + t_{real}) \tag{7.1}$$

Table 7.4 shows the processing time for both VLAN and UDP tunneling. As can be observed the processing for UDP sockets takes twice the time required for VLAN. However, we are talking about really small performance decrease introduced by usage of UDP tunnels.

Environment	Mean value	Standard deviation
Distr. IMUNES (VLAN)	0.0156	0.0069
Distr. IMUNES (UDP tunnel)	0.0320	0.0055

Table 7.4 Processing time for VLAN and UDP tunnels per one packet measured in ms

Chapter 8

Conclusion and future work

In the era of internetworking, most of the existing systems tend to be globally distributed. To measure the performance or simply test these systems network simulators that have a larger capacity are required. In order to suit this incrementing needs, but at the same time keep the model of the system as close to the real system, distributed network emulators are required. Network emulation of large networks gives an invaluable insight of the protocol performance on a large scale.

Standard testbeds offer to few resources for the cost of implementation and management. Network simulators offer scalability, with the price of neglecting details. Network emulators fill this gap and offer a very realistic environment while keeping the scalability. One of these emulators is IMUNES. Because the way IMUNES was built, the scalability of IMUNES outgrows most of network emulators ([26], [5]) and still keeps most of the details of the simulation. For making this emulation facility available even for larger networks, a distributed version is required.

In this thesis a way of building a distributed version of IMUNES was presented. This process started with a novel algorithm for large topology generation. Next, the performance of two different graph partitioning algorithms was compared, and the implementation of support for the better one was built in our system. In the end, the distributed version of IMUNES was built, based on decentralized architecture and the principles of scalability and transparency. The resulting system was deployed over a local area network. The resulting system is shown to be scalable and to put additional delay overhead of 0.05 ms per packet for simulated links that are deployed over a physical link.

In the future more attention should be given to server management, user man-

agement, experiment management and traffic management utilities. In distributed systems there are structures that can be adopted for distributed emulator purposes [34].

There are many areas where this system can be useful. In educational environment, students can learn about networks having a FreeBSD system that runs the simulation on the University and a variable OS for student client applications. As IMUNES offers support for all the protocols that reside on top of TCP/IP suite, this system can be used as a general testbed supplement for testing and development of new protocols as well as research of the existing ones. It can also be useful in development of new distributed applications. Since the code of the application does not need to be modified after it was emulated on IMUNES, after successful testing of protocols, they can be easily deployed on the real system.

Chapter 9

Abbreviations

AS - Autonomous System

BER - Bit Error Rate

CPU - Central Processing Unit

GUI - Graphical User Interface

IMUNES - Integrated multiprotocol network simulator/emulator

IP - Internet Protocol

LAN - Local Area Network

MTU - Maximum Transmission Unit

RAM - Random Access Memory

RPC - Remote Procedure Call

RTT - Round Trip Time

SSL - Secure Sockets Layer

TCP - Transmission Control Protocol

TSL - Transport Secure Layer

UDP - User Datagram Protocol

VLAN - Virtual LAN

References

1. David Alderson and Walter Willinger. A contrasting look at self-organization in the internet and next-generation communication networks. *IEEE Communications Magazine*, 43:94–100, 2005.

2. S. Lin B. Kernighan. An efficient heuristic procedure for partitioning graphs. volume 29, pages 291–308, 1970.

3. Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Rajive Bagrodia, and Mario Gerla. Glomosim: A scalable network simulation environment. 1999.

4. Albert-László Barabási and Eric Bonaneau. Scale-free networks. *Scientific American*, 288(5):50–59, 2003.

5. Micah Beck, Terry Moore, and James S. Plank. An end-to-end approach to globally scalable programmable networking. In *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 328–339, 2003.

6. Ken Jones Brent Welch and Jeffrey Hobbs. *Practical Programming in Tcl and Tk*. Prentice Hall PTR, 4 edition, 2003.

7. Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

8. Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, 1997.

9. Thomas T. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 1990.

10. Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262. ACM Press, 1999.

11. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th conference on Design automation*, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

12. FreeBSD. www.freebsd.org.

13. Richard M. Fujimoto, Kalyan Perumalla, Alfred Park, Hao Wu, Mostafa H. Ammar, and George F. Riley. Large-scale network simulation: How big? how fast? *mascots*, 00:116, 2003.

14. Jean Dollimore George Coulouris and Tim Kindberg. *Distributed systems: concepts and design*. Addison-Wesley, 2005.

15. Shashi Guruprasad, Robert Ricci, and Jay Lepreau. Integrated network experimentation using simulation and emulation. In *TRIDENTCOM '05: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05)*, pages 204–212, Washington, DC, USA, 2005. IEEE Computer Society.

16. IANA Internet Assigned Numbers Authority. www.iana.org.

17. Xuxian Jiang and Dongyan Xu. vbet: a vm-based emulation testbed. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 95–104. ACM Press, 2003.

18. P. Jr and F. Lamour. On the validity of a front-oriented approach to partitioning large sparse graphs with a connectivity constraint. In *Technical Report 94-37, University of California at Los Angeles, December 1994.*, 1994.

19. George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

20. Samson Lee, John Leaney, Tim O'Neill, and Mark Hunter. Performance benchmark of a parallel and distributed network simulator. In *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 101–108, Washington, DC, USA, 2005. IEEE Computer Society.

21. X. Liu, H. Xia, and A. Chien. Network emulation tools for modeling grid behaviors, 2003.

22. Damien Magoni. nem: A software for network topology analysis and modeling. *10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, pages 364–, 2002.

23. Harpal Maini, Kishan Mehrotra, Chilukuri Mohan, and Sanjay Ranka. Genetic algorithms for graph partitioning and incremental graph partitioning. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 449–457, New York, NY, USA, 1994. ACM Press.

24. METIS. glaros.dtc.umn.edu/gkhome/metis/metis/overview.

25. M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

26. L. Ni and P. Zheng. Empower: A network emulator for wireline and wireless networks, 2003.

27. ns2 network simulatior. http://www.isi.edu/nsnam/ns/.

28. Christoper R. Palmer and J. Gregory Steffan. Generating network topologies that obey power laws. *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE*, pages 434–438, 2000.

29. Rubn Mondjar Jordi Pujol Helio Tejedor Pedro Garca, Carles Pairot and Robert Rallo. Planetsim: A new overlay network simulation framework. *Lecture Notes in Computer Science*, 3437:123–136, 2005.

30. Larry L. Peterson and Bruce S. Davie. *Computer Networks: A System Approach*. Morgan Kaufmann Publishers, 2003.

31. Alex Pothen, Horst D. Simon, and Kan-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

32. Zrinka Puljiz and Miljenko Mikuc. Clustering network simulation: Graph partitioning approach. *ConTEL 2005*, 2005.

33. Zrinka Puljiz and Miljenko Mikuc. Distributed network emulator based on imunes. *SoftCOM 2006*, 2006.

34. Zrinka Puljiz and Miljenko Mikuc. Design issues in building a scalable network simulation/emulation middleware. *SoftCOM 2007*, 2007.

35. Zrinka Puljiz and Miljenko Mikuc. A hierarchical approach to generating power law internet-like topologies. *SoftCOM 2007*, 2007.

36. quagga. www.quagga.net.

37. Robert Ricci, Chris Alfeld, and Jay Lepreau. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev.*, 33(2):65–81, 2003.

38. Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350. Springer-Verlag, 2001.

39. Hawoong Jeong Soon-Hyung Yook and Albert-László Barabási. Modeling the internet's large-scale topology. *Proceedings of National Academy of Science*, 99:13382–13386, 2002.

40. W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Professional, 1994.

41. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

42. VMware. www.vmware.com.

43. S. Y. Wang, C. L. Chou, C. H. Huang, C. C. Hwang, Z. M. Yang, C. C. Chiou, and C. C. Lin. The design and implementation of the nctuns 1.0 network simulator. *Comput. Networks*, 42(2):175–197, 2003.

44. Xiao Fan Wang and Guanrong Chen. Complex networks: small-world, scale-free and beyond. *IEEE Circuits and Systems Magazine*, 3:6–20, 2003.

45. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.

46. Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6:1617–1622, 1988.

47. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

48. Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator.

49. xorp. www.xorp.org.

50. Ken Yocum, Ethan Eade, Julius Degesys, David Becker, Jeff Chase, and Amin Vahdat. Toward scaling network emulation using topology partitioning. *mascots*, 00:242, 2003.

51. Ken Yocum, Kevin Walsh, Amin Vahdat, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGCOMM Comput. Commun. Rev.*, 32(3):28–28, 2002.

52. M. Zec and M. Mikuc. Operating system support for integrated network emulation in imunes. *Proceedings of the 1st OASIS workshop (held along with ASPLOS-XI)*, 2004.

53. Marko Zec and Miljenko Mikuc. Real-time ip network simulation at gigabit data rates. *ConTEL*, 2003.

ABSTRACT

IMUNES network emulator is an excellent replacement for general purpose testbed, since it decreases the cost of equipment and management while increasing the scalability. However, IMUNES works on only one machine, so it has a limit in scalability, disallowing large-scale networks to be emulated. To circumvent this problem in this thesis we describe a distributed network simulator based on IMUNES. The distributed simulator is based on decentralized architecture, and thus avoiding the bottleneck. In addition, a novel algorithm for creating Internet-like topologies was designed, and the usability of two different graph partitioning algorithms was tested. The whole system was designed, implemented and evaluated. The results show that a decentralized version of distributed IMUNES is feasible. Moreover, the resulting system can be built in a scalable, transparent and secure manner while keeping the benefits of IMUNES.

Keywords:
network simulator, network emulator, IMUNES, distributed system, decentralized architecture, network performance, Internet, scalability, transparency, security, network topology, Internet-like networks, scale-free networks, graph partitioning

SAŽETAK

Mrežni emulator IMUNES predstavlja izvrsnu zamjenu za testne mreže opće namjene. On reducira troškove opreme i upravljana mrežom, a povećava skalabilnost sustava. IMUNES radi samo na jednom računalu, što znači da za emulaciju velikih mreža nema dovoljno resursa. Kao rješenje tog problema u ovome radu opisan je distribuirana inačica IMUNES-a. Distribuirana inačica temelji se na decentraliziranoj arhitekturi izbjegavajući centralni element koji bi bio usko grlo sustava. Osim distribuirane inačice IMUNES-a u radu je opisan novi algoritam za stvaranja topologije Internet tipa, te su uspoređene performanse dva algoritma za particioniranje grafova. Cijeli sustav je dizajniran, implementiran te evaluiran. Rezultati pokazuju da je decentralizirana inačica IMUNES-a izvediva, te da je ona skalabilna, transparentna i sigurna dok čuva pozitivne karakteristike IMUNES-a.

Ključne riječi:
Mrežni simulator, mrežni emulator, IMUNES, distribuirani sustav, decentralizirana arhitektura, mrežne performanse, Internet, skalabilnost, transparentnost, sigurnost, mrežna topologija, Internet tip topologije, scale-free mreže, particioniranje grafova

Resume

I finished the high school Lucijan Vranjanin in Zagreb by the year 1999. After enrolling in an undergraduate Faculty of electrical engineering and computing, University of Zagreb, I chose the field of Telecommunications and Informatics and I graduated from this field in 2004. During my undergraduate degree I received the national stipend. In 2004 I got enrolled into a graduate studies (Masters Studies) in the filed of electrical engineering, as well as into another undergraduate study in the field of computing. received the diploma from Computing studies in 2005 year. By the end of 2006 through and exchange program I visited Carnegie Mellon University, Pittsburgh, PA, USA, for four months while working in the area of wireless communications. During my masters studies the focus of my research was in the area of distributed network simulators and in that area I published four conference papers published on IEEE conferences.

Životopis

Završila sam srednju školu Lucijan Vranjanin u Zagrebu 1999 godine. Nakon upisa na Fakultet Elektrotehnike i Računarstva, Sveučilišta u Zagrebu, odabrala sam smjer Telekomunikacije i Informatika na kome sam diplomirala 2004. godine. Tokom 5 godina studiranja primala sam državnu stipendiju. 2004. godine upisala sam poslijediplomski studij, usmjerenja Elektrotehnika, te dodiplomski smjer Računarstvo na istom fakultetu, te ga završila 2005 godine. Krajem 2006. godine otišla sam na 4 mjeseca međunarodnog usavršavanja na Sveučilište Carnegie Mellon u Pittsburgu, PA, SAD. Tokom magistarskog studija fokus mojih istraživanja bio je u području distribuiranih mrežnih simulatora u kojem sam objavila 4 rada na IEEE konferencijama, te u području bežičnih mreža.