Prediction of Internet Characteristics for Distributed Applications

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Matthias Scheidegger

von Sumiswald

Leiter der Arbeit:

Prof. Dr. T. Braun

Institut für Informatik und angewandte Mathematik

Prediction of Internet Characteristics for Distributed Applications

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Matthias Scheidegger

von Sumiswald

Leiter der Arbeit:

Prof. Dr. T. Braun

Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät
angenommen.

Bern, 2. Februar 2007

Der Dekan
Prof. Dr. P. Messerli

Preface

The work presented in this thesis was performed during my employment as research and lecture assistant at the Institute of Computer Science and Applied Mathematics (IAM) of the University of Bern. The research was conducted within two research projects, the European Union project IST-Intermon and the Swiss National Science Foundation project XBAC.

I would like to thank everybody who has supported me during my stay at the Computer Networks and Distributed Systems group (RVS), especially Prof. Dr. Torsten Braun for supervising the work and for his guidance and advice.

I would also like to thank Prof. Dr. Klaus Wehrle for having accepted to read and judge this work, and Prof. Dr. Horst Bunke who was willing to be the co-examinator of this work.

Furthermore, many thanks go to my colleagues at the IAM for numerous fruitful discussions and for creating a very agreeable atmosphere. Special thanks go to Florian Baumgartner, Dragan Milic, Marc Brogle, Markus Wälchli, Thomas Bernoulli, Thomas Staub, Gerald Wagenknecht, and Markus Wulff. Many thanks go to Benjamin Zahler who worked with me and helped with developing and implementing. I would also like to thank our system administrator Peppo Brambilla and our secretary Ruth Bestgen for their excellent work.

I am deeply grateful to my relatives and friends Ursula Scheidegger, Bernhard Scheidegger, Andrea Schwab, Annina Kienholz, Marc Hugi, and Tobias Roth for all their support and for having a great time together. Finally, I thank Anna Mund for her support and understanding and for sharing a wonderful time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

The global Internet has grown at an amazing speed over the last decades. What once started with only a few interconnected computers is now a immense network connecting hundreds of millions of computers. This tremendous growth comes not without problems. The size and complexity of the Internet makes its behavior obscure and hard to foresee. New protocols and distributed applications may perform well in small-scale tests and simulations but still exhibit unexpected problems when run on the Internet, and changes to the network topology may have unwanted side effects.

This problem can be approached from two angles. On the one hand, if a distributed application requires a static deployment that cannot be easily changed afterwards, sophisticated simulation techniques may be an adequate tool for predicting the performance of a given deployment scenario. On the other hand, many recent applications use peer-to-peer designs that adapt dynamically to changing network conditions in order to gain efficiency and robustness. They use quality of service measurements of end-to-end paths to find optimal overlay topologies. Planning and elaborate simulations are of limited use to these applications. Instead, accurate and efficient prediction of end-to-end quality of service is a major factor for their performance. In this work we focus on both, prediction of application behavior through simulation and the prediction of end-to-end quality of service to support the adaptivity of distributed applications.

Simulation is a traditional way to assess the viability of a distributed application before the actual deployment. Unfortunately, conventional packet-based simulation tools are not capable of simulating even reasonably sized parts of the Internet due to its complexity. One problem is that the simulation of Internet scenarios requires very high amounts of memory and computation time. Parallel simulators running on high-performance computers reduce the computation times considerably and often have enough memory to store the simulation state even for large scenarios. But even then, it is very hard to create accurate simulation scenarios. Routers may be configured in many different ways, and the traffic patterns in the Internet are very complex. Unlike telephony networks, the Internet carries traffic generated by thousands of different applications with varying characteristics. Creating fine-grained scenarios on the packet level would

be a time-consuming endeavor. Moreover, information about the exact network topology, router configurations, and traffic patterns is not generally available. The Internet consists of a vast number of independent administrative domains, and network administrators do not normally publish details about their networks. Therefore, Internet simulations often model the network on a high abstraction level that hides unneeded details. The parameters of these models can often be determined by making measurements on the Internet and do not require detailed knowledge of the network. Using simpler models also leads to much faster simulation. Most models are designed to preserve the details of one aspect of the simulation at the cost of reduced accuracy of the other aspects. Hence, they must be carefully selected to fit a given scenario.

Part of the reason for the Internet's success is the ability of many protocols to operate with very little support from the network itself. The widely used Transport Control Protocol (TCP) for example only relies on the network's capability to route Internet Protocol (IP) packets to their destination. Packet loss and network congestion are detected and compensated by mechanisms located at the communication endpoints. This focus on end-to-end mechanisms makes the Internet flexible towards growth and changes. Unlike TCP, whose adaptivity is limited to the network path between two communication endpoints, peer-to-peer networks and other distributed applications may employ more sophisticated schemes to adjust to changing network conditions. Peer-to-peer networks are made up of a large number of peers, interconnected by a network of logical links with no direct relation to the structure of the underlying physical network. The topology of these links can be dynamically changed to maximize robustness and efficiency. However, the construction of optimal peer-to-peer topologies is far from trivial and usually application dependent. One of the basic challenges is how to predict the characteristics of a network path between two peers. Large-scale measurements may not be practicable and should be replaced by more efficient approaches. Various solutions to this distance estimation problem have been proposed in the literature.

1.2 Contributions

In a first part of this work we address the aspect of predicting Internet behavior through simulation using high-level abstraction. There are many different abstract models, each one tailored to a specific purpose or type of network. Fluid simulation for example aggregates packets into fluid flows in order to save the effort of modeling each of them separately, with the consequence of making TCP simulation less accurate due to TCP's sensitivity to the exact timing of packets and the order of packet losses. Other approaches simplify the modeled topology at the risk of overlooking possible causes for congestion in the network. Internet scenarios often consist of several parts, each modeled best with a different abstraction. When we simulate a distributed application deployed at several sites for example we may want to simulate the local networks of the sites in high detail while modeling the Internet core with more coarse-grained methods.

We have designed and implemented a hybrid simulator aimed at this kind of scenario. It is based on an extension mechanism to the packet-based ns-2 simulator that allows to insert plug-in modules into a simulation scenario. Each of these modules may simulate part of the modeled network using its own

abstraction. We have used this extension to combine packet-based simulation with an abstract network model for the scalable simulation of inter-domain networks and large inter-domain traffic aggregates. This network model includes concepts from time stepped fluid simulation and analytical queuing theory. The extension mechanism can also be used to include other types of modules into the same scenario. Connected through an ns-2 topology, they form a simulation scenario with multiple combined abstractions.

The hybrid simulator has been integrated into a large, distributed architecture for network monitoring, modeling, simulation, and visualization, primarily aimed at users such as network administrators. The architecture provides seamless integration of tools for the various tasks in a single graphical user interface. For example, a user can schedule measurements on distributed monitors in the network and, once available, run a simulation scenario based on the resulting topology and traffic data as well as user-made changes. The simulator output can then be sent to the visualization component to generate graphical output.

In a second part of the work we concentrate on predicting Internet behavior from an end-to-end perspective. The existing approaches to this distance estimation problem can be roughly divided into two categories:

- clustering approaches, that cluster endpoints in order to reduce the number of necessary end-to-end measurements
- coordinates-based approaches, that map endpoints to a coordinate space and use that space's distance function to estimate network distances.

Clustering approaches often rely on dedicated infrastructure in the Internet, which is a hurdle for their deployment. Coordinates-based approaches can only estimate distances between members of the same peer-to-peer system since independent hosts do not know about each other's coordinate system. Furthermore, many systems from both categories require a certain minimal size to perform well, which prohibits local deployment.

Another common problem of distance estimation systems is that they focus on long term average distances of a single type even though Internet characteristics are diverse and may change rather rapidly over time. Applications would benefit from additional information like variances or trends of end-to-end distances. Unfortunately, the computation of more sophisticated predictions is also more demanding on the computers running the system. Furthermore, while conventional distance estimation services can operate with very few measurements per endpoint pair such predictions generally require larger series of measurements.

We propose a service that provides more sophisticated predictions of end-to-end network characteristics than common distance estimation approaches because it considers series of distance measurements instead of single, averaged values. In this system, nodes from the same locations in the network organize in groups. Inside these groups, the nodes monitor application traffic to obtain end-to-end measurements and store the results in a peer-to-peer repository. Based on the data in this repository the service can then make distance predictions. Since the amount of measurement data in the repository may be large we employ a clustering algorithm to combine different measurements into a single one where possible. The peer-to-peer design used in the service distributes the storage of measurement data and the computation of predictions among multiple

peers. It also makes additional infrastructure unnecessary. Another desirable property of the service is that it does not require a large-scale deployment to be useful. Its peer-to-peer groups may operate in isolation and can provide distance predictions even if deployed only at a single site.

Our main contributions in this thesis are the following:

- We have developed a hybrid simulator that combines fine-grained packet-based simulation with a more coarse-grained but efficient inter-domain network model. This allows for scenarios that would not be possible to simulate with conventional tools. The hybrid simulator is mainly useful for predicting the behavior of distributed applications deployed at several sites in the Internet. It can employ fine-grained models to simulate the application itself while using a coarse-grained inter-domain model for network between the sites.

- In addition to the hybrid simulator, we have implemented a system for the automatic creation of measurement-based simulation scenarios, scheduling of simulation jobs, and processing of simulator output. This system was used as link to integrate the hybrid simulator into a large architecture for the monitoring, simulation, and visualization of inter-domain networks.

- We have created a quality of service prediction architecture that is aimed at helping dynamic distributed applications like peer-to-peer networks to adapt to the quality of service on the underlying physical network. Unlike other approaches, our service is able to provide sophisticated predictions for arbitrary types of end-to-end measurements, and it does not require a large deployment to be useful. Furthermore, it can operate without any additional infrastructure in the network due to its peer-to-peer design.

1.3 Thesis Outline

Chapter 2 gives an overview of available modeling and simulation techniques with special focus on their suitability for large-scale Internet scenarios. We discuss the shortcomings of sequential packet-based simulation and discuss two main directions of research, parallel simulation and abstraction.

In Chapter 3 we point out the importance of end-to-end adaptability for protocols and architectures in the Internet, having a closer look at peer-to-peer networks. The second part of the chapter is concerned with systems supporting the adaptability of distributed applications, such as end-to-end distance estimation services. We state several factors for the usability of these distance estimation services and discuss the shortcomings of the existing approaches concerning their ease of deployment, their reliance on additional infrastructure in the network, or the scope of their estimates.

We present a scalable simulator for inter-domain scenarios in Chapter 4. Its basic idea is to combine the packet-based simulation approach with high-level abstractions of the network to exploit the advantages of each in a single scenario. We describe a high-level model specifically aimed at inter-domain networks, combining elements of queuing theory and time stepped fluid simulation. We also present the integrated of this hybrid simulator into a toolkit for distributed monitoring and simulation of inter-domain networks.

Chapter 5 we propose a peer-to-peer-based distance prediction service that has several beneficial properties. Unlike other approaches, it can provide quality of service predictions for arbitrary hosts in the Internet even if the service is deployed only at a single site. Moreover, it is able to give predictions of future quality of service instead of the simple estimates average round-trip time commonplace in other schemes. We present the service's distributed architecture and the clustering algorithms that are key elements for the viability of the approach.

Finally in Chapter 6, we conclude the thesis by summarizing the results and contributions.

Chapter 2

Network Modeling and Simulation

Because computer networks are highly complex systems, designing and controlling them is not trivial. It requires tools to study and evaluate possible alternatives, and to validate the design before building a network. For example, before laying a cable an Internet service provider would most likely want to quantify the potential benefit of the new network link. Also the deployment of a distributed application that requires additional network infrastructure shouldn't be made without prior analysis.

Creating tools for predicting the behavior of computer networks and distributed applications has been a focus of research for many years. Discrete event systems (DES) have taken a predominant role in this area because they provide a natural way to model computer networks. Events occur when a packet arrives at a computer or when a network link fails, for example. However, there is no single, commonly accepted DES model for computer networks. Depending on the goal of the analysis, there are many different approaches to model and analyze a computer network with discrete event systems. Section 2.1 gives an overview of possible models for networks and network components.

One approach to studying computer networks is to analyze these models using mathematical means. This method, called analytical modeling, has the advantage that it yields closed formulas for many performance measures, which make dependencies between parameters and the performance of the network clearly visible. Unfortunately, the structure of real networks is usually too complex to be modeled by analytical means. The assumptions made by such models often tend to oversimplify the behavior of a real network. Many computer network researchers have thus turned to simulation. In cases where analytical models are not tractable, simulation often provides sufficiently accurate numerical approximations of the behavior of network models. Simulations are comparable to laboratory experiments. They mimic the behavior of a real network and compute its reaction to certain events. By running simulations with many small variations (usually generated by random number generators) we can determine statistical characteristics such as the average and variance of measures of interest. However, because of their stochastic nature, simulations can only provide estimates.

Simulations are able to model much larger and more complicated computer networks than analytical models. Nonetheless, a fundamental problem of simulation is that the larger the scenario, the more CPU power it takes to compute the results. In fact, today's Internet cannot be faithfully simulated by any known method. A large body of work has been devoted to this issue. Section 2.2 gives an overview of the numerous methods proposed to alleviate this scalability problem.

2.1 Modeling

When we want to study a computer network we can either build a physical testbed and observe how it performs, or we can create an abstract model of the network and try to gain insights by analyzing it. Both approaches have their merit. Testbeds are generally very costly to build and hard to change. Models are less realistic than testbeds, but they are also cheap to build and usually easy to change. In the following we will take a brief look at queuing theory, a well-known approach for building analytical models of computer networks and other discrete event systems. We will have a look at traffic modeling.

2.1.1 Queuing Theory

Queuing systems are a popular and well established class of discrete event systems. They are particularly well-suited for computer networks because of the importance of queues in network components such as routers. A basic queuing system consists of a customer (or job) arrival process, one or several servers that process customer requests, and a queue where customers wait until they can be served (see Figure 2.1).

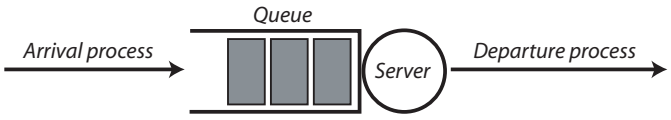


Figure 2.1: A basic queuing system

Several aspects of a queuing system need to be specified before we can analyze it. A common way to classify these aspects is Kendall's notation $A/B/c/K/m/Z$, where

A – specifies the arrival process, i.e. the distribution of the time between the arrival of two consecutive customers (called inter-arrival time).

B – specifies the service time distribution.

c – is the number of (identical) servers in the service station.

K – is the system's capacity, i.e. the capacity of the queue plus the capacity of the service station. If omitted, this parameter is assumed to be infinity.

m – is the capacity of the source population. This is commonly set to infinity.

Z – specifies the system's queuing discipline, which dictates the order in which customers are served. Common choices are FCFS (First-Come-First-Serve, also known as FIFO), LCFS (Last-Come-First-Serve), RSS (Random-Selection-for-Service), or PRI (priority scheduling).

Standard choices for A and B are shown in Table 2.1. Based on the queuing system specification, we can calculate numerous performance measures. Examples are the utilization of the system, the mean sojourn time of a customer (i.e. the time between the arrival and the departure of a customer), and the mean number of customers in the queue. Another important property is the state of the system, often described using the state probabilities p_n. p_n describes the probability that n customers are in the system at a given time.

Table 2.1: Standard stochastic processes for queuing systems

M	Markovian/Poisson arrivals (exponentially distributed inter-arrival times)
D	Deterministic (constant)
H	Hyper-exponential (linear combination of exponential distributions)
E_k	Erlang-k arrivals.
PH	Phase type arrivals.
G or GI	General (independent) distribution, i.e. the distribution is not a priori known.

The abstraction of a queuing system can be easily applied to networking. Here, a "customer" is usually a packet or cell arriving at a router. Arrival processes model applications generating traffic. When a router receives a packet it determines on which outgoing port it should be forwarded. If the corresponding network interface (server) is busy, the packet will be sent to the queue according to the queuing discipline. A powerful property of queuing theory is that queuing systems like the one in Figure 2.1 can be connected to form a network. However, solving the equations necessary for a large-scale system of queues may quickly become very time consuming. Moreover, many arrival processes that lend themselves to queuing systems because of their simplicity are rather bad approximations of packet arrival processes in a real network. For example, Markovian arrivals lead to simple formulas, and they may be an acceptable approximation of arrivals in highly controlled networks such as telephony networks, but they are usually unsuitable for modeling packet arrivals in the Internet. We examine this problem in more detail in the next section.

2.1.2 Traffic Modeling

One of the most important factors for accurate prediction of Internet behavior are traffic models. Network elements like switches, routers, and links are relatively easy to model accurately. However, the traffic generated by the end systems in the network may exhibit arbitrarily complex patterns. This is due to the fact that this traffic is normally triggered by human interaction (e.g. a

person clicking on a link in a web browser, or downloading a file via FTP). On the Internet, the multitude of utilized protocols with varying characteristics adds to this problem. Recent work on traffic models for queuing systems [62] suggests that such complicated arrival processes can be accurately modeled as Batch Markovian Arrival Processes (BMAPs). However, protocols like TCP, that adapt to network congestion using a feedback mechanism, remain hard to model analytically.

In the network core, the traffic arriving at a router is a mix of many application flows. It was long assumed that the bursty characteristics of the single flows would even out in the core of the network and thus allow for approximation with simple Poisson models. That assumption has proven to be wrong [90]. Wide-area traffic in the Internet shows self-similar properties. That means that its short-term variability does not "even out" over larger time-scales like traffic generated with a Poisson model would. This phenomenon has also been observed in World Wide Web traffic [24] and local area networks using ethernet [69].

Nevertheless, for many areas verified traffic models exist. A compilation can be found in [31]: User "session" arrivals are well-described using Poisson processes. Connection sizes and durations can often be modeled using log-normal distributions, whereas their extremes may be better modeled using heavy-tailed distributions such as Pareto. Danzig et al. describe an empirical TCP workload model as observed at network stubs [27] specifically aimed at wide-area TCP/IP network simulations. Analytical models for various TCP connection types such as Telnet, NNTP, SMTP, and FTP are described in [89]. Because of the dominance of web traffic on the Internet, special attention has been given to creating models of this type of traffic. Mah [74] has analyzed packet traces of HTTP conversations to formulate an empirical model for HTTP network traffic. In [25], execution traces of the somewhat aged NCSA Mosaic web browser have been analyzed to gain insight into the characteristics of WWW clients. The Scalable URL Reference Generator (SURGE) [7] also applies analytical models to web traffic, but instead of including these models in a simulator the authors use them to drive a traffic generator for physical testbeds.

2.2 Simulation

While analytical models of computer networks have the advantage that they provide closed formulas for a given problem, they are also hard to come by. Their complexity makes them tractable only for small scenarios. Moreover, they often rely on oversimplifying assumptions such as Poisson packet arrivals, which further reduces their usefulness. Consequently, a large part of current network research relies on simulations instead of analytical modeling.

Like analytical models, network simulations normally rely on discrete event systems. However, each simulation run only computes a single outcome the scenario may have, called *sample path* or *state trajectory*. In order to obtain statistically meaningful results, simulations have to be run several times with slightly different parameters. This "noise" is created by so-called random number generators (RNGs), which mimic true random processes using deterministic

tools (Several algorithms for random number generation can be found in [101]). Once we have a sufficient number of simulation results we can give good estimates for the behavior of the network.

The most common simulation scheme is sequential *event scheduling*. Simulations start with a given state of the network and a virtual simulation time that is usually set to zero. A queue holds the events that have not been executed yet, sorted by their scheduled activation time. We schedule an event by inserting it into this queue at the correct position. Some events are inserted before the simulation starts. These commonly trigger network components such as traffic sources, which in turn schedule more events. Many simulators define a special stop event that ends the simulation. The simulator starts by dequeuing the event with the smallest scheduled time. Then, it sets the simulation time to that of the event and executes it. The event may change the network state and schedule additional events. When the event has been processed, the simulator repeats the procedure with the next event and continues to do so until either no events are left or it encounters the stop event. This way of simulating discrete event systems is natural for modeling network components such as routers, since they are usually implemented in an event-driven fashion themselves.

However, for some applications the event scheduling approach is unnatural and makes implementing simulation models unnecessarily hard. An approach that may provide a better environment in these cases is the *process-oriented* simulation scheme. Here, simulation entities can be formulated as processes that resemble the control flow of normal computer programs, with the addition that a process may also wait for events. For example, a simulated network client might initiate a connection to a peer, wait until the connection is set up, then send a message to the peer, wait for a response, and finally terminate the connection. This alternate way of simulating discrete event systems can be efficiently implemented using user-level threads, which are available on most current operating systems. It is well suited for simulating the interaction of client programs. Nevertheless, most network simulators implement the event scheduling scheme.

An important aspect of every simulator is the granularity of its network model. A popular approach, packet-based simulation, simulates networks at the level of packets and above, but does not explicitly consider lower-level details like the transmission of frames over ethernet links. While this abstraction is suitable for a wide range of simulation scenarios it also suffers from severe scalability problems when applied to large-scale networks. This problem is being approached by two main directions of research, parallel simulation, and finding alternative abstractions. In Section 2.2.1 we look at the packet-based approach and its scalability problems, and we discuss parallel simulation and abstraction approaches in Sections 2.2.2 and 2.2.3, respectively.

2.2.1 Packet-Based Simulation

When designing a network simulator we have to decide about the level of detail the simulator should support. There is a tradeoff between the level of detail of the network model and the time it takes to run a simulation. Fine-grained network models considering physical details of the network would be preferable because they produce more accurate results than coarse-grained ones, but they are impractical since they can only simulate very small networks within an ac-

ceptable amount of time. A rather popular compromise is to simulate networks at the packet-level. This level of detail is sufficient for studying a wide range of scenarios, including new protocols, routing strategies, and queuing disciplines.

The ns-2 Simulator

The sequential packet-based ns-2 simulator [12] has seen widespread use in the research community for several years. It has been used to investigate protocols like the Transmission Control Protocol (TCP) and develop new queuing disciplines like Random Early Detection (RED). ns-2 provides an object-oriented split-programming environment: The event engine and all time-critical functionality is implemented in the C++ programming language for efficiency. All functionality required to configure and control a simulation is available through an object-oriented variant of the Tcl programming language (OTcl). This ensures relative ease of use without noticeably slowing the simulator down. Nevertheless, this also results in a steep learning curve for researchers who need to extend the simulator. Since we base the implementation of our hybrid simulator described in Chapter 4 on ns-2 we will take a closer look at ns-2's design.

The ns-2 simulator models network topologies as a graph using three basic building blocks: links, nodes, and agents. *Links* are the edges of the graph and model the effects of packet transmission through physical links. They can have several user-specified attributes such as bandwidth, propagation delay, or bit error rate. A peculiarity of ns-2 is that links also manage packet queues, although in real networks this would be done by the nodes. Ns-2 *nodes* are the vertices of the graph and represent physical machines in the network, such as workstations or routers. Accordingly, their main tasks are to forward packets from one link to the next, and to host user applications (called agents in ns-2). Ns-2 does not model any processing delay in the nodes. Thus, the simulation of network characteristics such as delay and packet loss depends solely on the links. *Agents* model user applications. They are the traffic sources and sinks in a simulation scenario. Like in real workstations the agents use a port number to distinguish themselves from other agents on the same node. As we will see in Chapter 4, our hybrid simulator several modifications to the ns-2 code, especially to the ns-2 node. The original structure of an ns-2 node is shown in Figure 2.2.

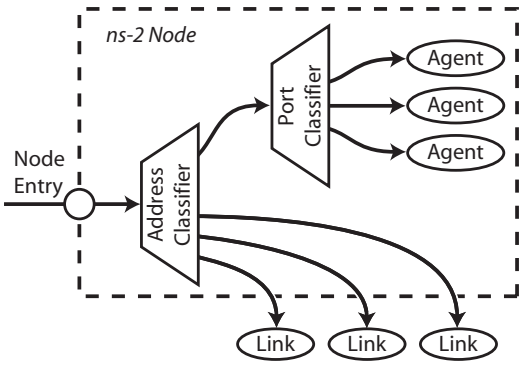


Figure 2.2: The structure of an ns-2 node

We can see that packets arriving at the node directly go into the address classifier, which is the routing component of the node. It chooses the next link for a packet depending on its destination address and hands it over to that link for further transmission. However, if a packet has reached its destination node the address classifier does not forward it any further but hands it over to the port classifier. The agent responsible for the packet's port number will then receive the packet for processing.

Other Packet-Based Simulators

Opnet [17] is a commercial packet-based simulator. Unlike ns-2, it supports process-oriented simulation through a language called Proto-C. Moreover,its node models are made up of interconnected modules, which makes them much more configurable than their ns-2 counterparts. Being a commercial product, Opnet also comes with several tools to edit scenarios and analyze simulator output.

The INET package for the OMNeT++ [135] simulator framework is another example of packet-based simulation. It includes a variety of models for protocols used on the Internet such as IP (both, versions 4 and 6), TCP, UDP, and ICMP. Furthermore, models for network components like routers, switches, and transmission media of various types are available. It also includes models for several kinds of applications and routing strategies. If desired, simulation scenarios can also incorporate details below the packet level such as ARP messages on ethernet links. The OMNeT++ framework and simulation kernel itself employs a modular design reminiscent of Opnet.

Scalability Issues of Sequential Packet-Based Simulators

Sequential packet-based simulation has proven useful for investigating various research problems. Nevertheless, the approach lacks scalability. The sheer number of events makes the simulation of large-scale Internet scenarios a time consuming endeavor. The execution time of a simulation is roughly proportional to the occurring number of events. Unfortunately, when we increase the size of the simulated network there are several factors resulting in an increasing number of events.

- Growing network size leads to a greater average distance between traffic sources and traffic sinks, making packets take more hops before they reach their destination. Every hop causes additional events, normally at least one per hop. In Internet-alike (small-world) topologies the average number of hops on a network path scales with $O\big(\log(N)\big)$ to the number of nodes N.

- Bigger scenarios normally also include a greater number of flows. It is reasonable to assume that the number of flows (and thus the number of transmitted packets) increases proportionally with the number of nodes, which results in a linear increase of the number of events.

- The number of end users with broadband Internet access is increasing, causing a higher demand for multimedia services like web radio or video

streaming. Consequently, in up-to-date simulation scenarios the bandwidth used per flow is rising as well, which again results in a linear increase of the number of events.

Combined, these effects result in a significant increase of the number of events a simulator has to process when studying large-scale scenarios. The exact scaling behavior of a simulator depends on many parameters. Given the considerations above, we can roughly estimate that the number of events in a scenario scales with

$$O\big(B \cdot N \cdot \log(N)\big) \quad , \tag{2.1}$$

where B is the average bandwidth used by a flow and N is the number of nodes in the scenario.

There are three independent ways of speeding up simulation (see Figure 2.3) [29]. Firstly, the implementation of the simulator can be optimized. For example, the event scheduling algorithm can be improved with techniques like calendar queues [14] or timing wheels [136] to have $O(1)$ complexity instead of the $O(\log n)$ complexity of naive implementations. In split-programming simulators like ns-2, parts of the simulator written in a scripting language (Tcl in ns-2) can be reimplemented using a lower level compiled language (C++ in ns-2). This optimization approach is the most straightforward but also the least effective. The improvement gained from code optimization is usually not very big. Moreover, most widely used simulators already use strongly optimized code.

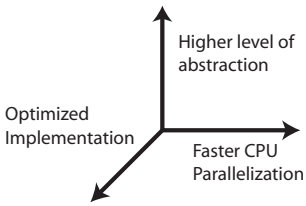


Figure 2.3: Three approaches to speeding up simulation

Another possibility to speed up simulation is to use more CPU power to run the simulator. With sequential event-based simulators this approach is very limited because we can only replace a single CPU, which makes the simulation two or three times faster but cannot be improved further. Parallelization overcomes this obstacle. Parallel simulators can run on many CPUs at the same time and thus scale much better to large-scale scenarios. They are especially suited for super computers with a large number of parallel CPUs. A large body of work has been devoted to parallel simulation.

A third way to make simulation of large-scale scenarios possible is to reduce the level of detail. Packet-based simulation is considered a good compromise between speed of computation and accuracy for many simulation scenarios. For large-scale networks however, it may be more efficient to use a more coarse-grained model of the network. Many different possible abstractions for large-scale simulations have been proposed in the literature. Section 2.2.3 gives an overview.

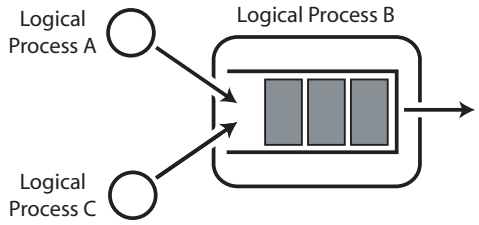


Figure 2.4: Two parallel logical processes using conservative synchronization

2.2.2 Parallelization

Most event-based simulations execute events one after the other. This approach is called *sequential* event-based simulation. While it is comparatively simple, it does not exploit the inherent concurrency of a computer network. Events often only influence their immediate vicinity in the simulated network topology. Thus, events in different parts of the topology could be simulated independently, which would allow for distributing the simulation to multiple CPUs. A similar effect can be found on the time axis. An event early in the simulation has a negligible effect on events much later in the simulation. The simulation approaches known as *parallel discrete event simulation* (PDES) and *time parallel simulation* stem from these two observations. They are also referred to as *time-division* and *space-division* approaches. Unfortunately, it is almost never possible to divide a scenario into completely independent parts. There are always events from one part that influence another. Since this necessitates expensive "fix up" computations it is important to find partitionings that minimize the number of such events.

Space-Parallel Methods

Space-parallel simulators divide the simulated network into partitions (e.g. single routers, subnetworks, etc.) and perform the computation for each on a separate processor. We call the independently running parts of the simulation *logical processes*. During execution, the virtual simulation time in each logical process is roughly the same. Nevertheless there may be differences, which leads to synchronization problems when the logical processes exchange events. Discrete event simulations require that events are executed in the order of their scheduled time. This is easy to do in a traditional sequential simulator. Events are stored in a queue sorted by their activation time, and whenever the simulator is ready to process a new event it simply dequeues the next event from it. Space-parallel simulations however use several queues, one for each logical process. Accordingly, they must be synchronized through some mechanism.

Conservative Synchronization A widely used solution to this problem is *conservative* synchronization. The logical processes communicate to find a safe interval of simulation time in which they can be sure to receive no events from other logical processes. This interval is called the *lookahead*. While it lasts the logical process can execute its local events without the need to handle any external events. When there are no local events left in the lookahead interval, events are exchanged between the logical processes and a new lookahead interval

must be negotiated. Note that the lookahead is not necessarily the same for all logical processes. A logical process' lookahead only depends on the events of another logical process if they in fact exchange events (for example, two adjacent routers in a network scenario). In the example in Figure 2.4, two traffic sources (logical processes A and C) send packets to a multiplexer component with a queue (logical process B). The lookahead interval of B is $[t, t')$, where t' is the time when a packet from either A or C arrives at B. B in turn affects the lookahead of any logical process that receives its packets.

Several parallel simulators implement the conservative approach (Nops [97], which is based on TeD [91], and SSFNet / DaSSF [23] to name a few). Conservative network simulation has also been used in [41] and [93]. A well-known algorithm based on process-oriented simulation is described in [79]. Logical processes send event messages to each other unless there are no events left to be sent. In that case they send *null messages* announcing the interval in which they will not send any further events. With this information the logical processes can compute their lookahead interval. Sending null messages protects the approach from starvation.

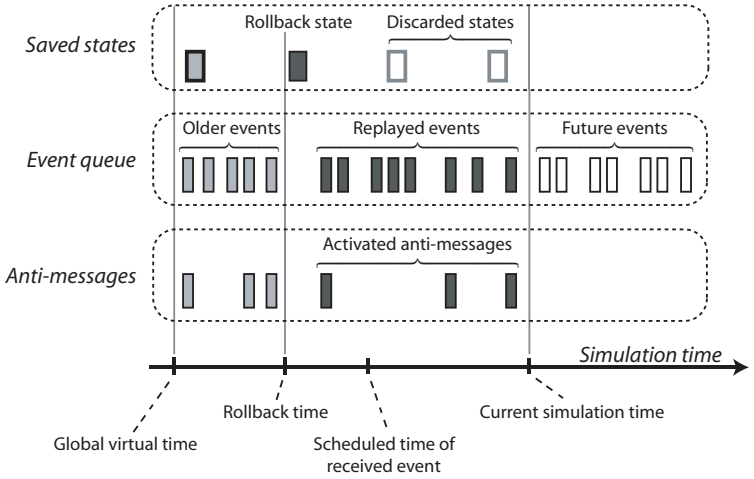


Figure 2.5: A rollback operation in the Time Warp algorithm

Optimistic Synchronization Jefferson [56] pioneered the *optimistic* approach to space-parallel simulation with the Time Warp algorithm. Unlike conservative methods, Time Warp lets logical processes execute events without waiting for external events. Consequently, an event may reach a logical process whose simulation time has passed the event's scheduled time. If this happens, the logical process performs a rollback to a known state just before the scheduled execution time of the received event, and recomputes the simulation from there. This scheme obviously requires the logical process to regularly save its state, typically once every few seconds of processing time. Furthermore, already processed events cannot be discarded because they might be used again for re-computation after a rollback. The Time Warp algorithm restricts the memory used for storing state and events by keeping track of the *global virtual time* (GVT), the lower bound of all active events in the system. If an event

was scheduled before the GVT it can be safely discarded. The same is true for saved states of logical processes from before the GVT. Hence, the amount of saved states and events does not grow indefinitely (Mattern discusses several efficient algorithms for GVT approximation in [76]). Another important aspect of the algorithm is the handling of the possible side effects of events. When a logical process receives an event scheduled in the past, it does not suffice to just rollback and recompute the relevant time interval. The events sent during this interval may have influenced the state of other logical processes. Worse, they may have caused other processes to send events of their own. This is solved by maintaining a sorted list of so-called *anti-messages*. For each message sent to another process, an anti-message is inserted into this list. If the logical process performs a rollback to time t, all anti-messages with a timestamp greater than t are sent to the appropriate receivers where they have two effects. First, they cancel out the original message, effectively removing it from the receiver's event queue. Second, if the receiver's simulation time has progressed beyond the scheduled time of the canceled event, the anti-message triggers a rollback. Thus, the rollback of a single logical process effects an optimal cascade of rollbacks in other logical processes. Figure 2.5 illustrates the steps during a rollback of a single logical process.

Several simulators based on the optimistic method have been developed. Telesim [134] uses Time Warp to simulate telephony networks. Parsec [5] and its descendant GloMoSim (Global Mobile system Simulator) [144] are well-known simulators originally based on the Maisie simulation language [6], which supports sequential as well as conservative and optimistic parallel simulations. Optimistic space-parallel simulation was also used in [110] for evaluating reliable multicast protocols. Hao et al. [46] have shown that Time Warp works better with small logical processes.

Federated Simulators Although parallel simulation has been known for long time, its adoption in industrial simulators has been sparse [84]. This has been attributed to the industry's reluctance to reimplement existing and validated simulation models. A popular approach to alleviate this problem is to *federate* existing sequential simulators, i.e. to coordinate several instances of the same simulator (or even different simulators) with a messaging back-end [78, 84, 107]. This technique has also applied to the ns-2 simulator described in Section 2.2.1 [108, 92].

Time-Parallel Methods

The alternative to partitioning simulation scenarios in space is to partition the duration of the simulation into subintervals. This can theoretically result in a high degree of parallelism because the time axis is continuous. Nonetheless, even on the time axis the problem remains that it is difficult to partition a simulation scenario into independent parts.

Chandy and Sherman's algorithm [16], which is in fact a combination of time-parallel and space-parallel approaches, comprises the basic approaches for time-parallel simulation. The simulation scenario is partitioned into space-time regions, which are simulated on independent processors. The simulation of each region starts with with a guessed initial state. When the final state of a region becomes known it is sent to the regions that depend on it. These

regions then correct their results (if necessary by re-simulating the whole region) and in turn send their final state to any dependent regions. This correction step is commonly referred to as a "fix up." The simulation stops when the initial states of the regions match the final states of their respective preceding regions. This approach can significantly speed up simulations, but only if the scenario is "nearly decomposable," i.e. communication between the regions is rare. Otherwise, the simulation may take even longer than sequential event based simulation. The space-time approach has also been used in Genesis [131]. The "fix up" in time-parallel simulation can be reduced if we can find points in time where the initial state has no influence anymore [40, 47, 3, 35, 140, 58, 71].

Time-parallel simulation can yield a significant amount of parallelism, and it may be used to parallelize elements of network simulation that could not be parallelized using space-division approaches (e.g. single queues). However, it is only applicable to special cases with relatively little state information. The state information of large-scale network scenarios is much too big to make time-parallel simulation viable.

2.2.3 Abstraction

The enormous computational cost for discrete event simulations of large-scale networks is due to the amount of events the simulator needs to process. Parallel simulation approaches take advantage of the increased computational power of parallel computers to process the large amount of events in a reasonable time. However, even when used in parallel simulators the packet-based approach may be inadequate for Internet scenarios. We are often missing the information for creating accurate packet-based models of the network. If we just guess at the topology and configuration of the network or the composition of network traffic we may obtain an accurate simulation of an unrealistic model of the network. In this section we look at the converse approach: simulators that use simplified, abstract models generating much fewer events while keeping a reasonable level of accuracy. Several different aspects of a network simulation scenario may influence the number of generated events; for example, the average number of packets per flow or the number of hops in a simulation topology. Abstraction methods simplify the simulation model with respect to one or more of these aspects, and they often achieve a significant reduction in the number of events. However, this speedup always come with a loss in simulation accuracy. Abstractions are normally aimed at a certain aspect of the network scenario and oversimplify other aspects in order to speed up the computation. Depending on the problem at hand we thus have to select a suitable abstraction.

One of the most well-known abstractions for network simulation is *fluid simulation*. This method models the large amount of packets in a high bandwidth flow as a homogeneous fluid flowing through the network links. A fluid flow does not distinguish single packets but only models the changes in bitrate over time. This eliminates the large amount of packet related events in the simulator. Events are only caused by changes in the bitrate of a a flow, either at its source or in a network router. Figure 2.6 shows an example of an on/off traffic source. Every on-period of the traffic source generates a number of packet arrival events – higher bitrates generate more packets. The fluid model however only generates events when the rate of the traffic source changes,resulting in a

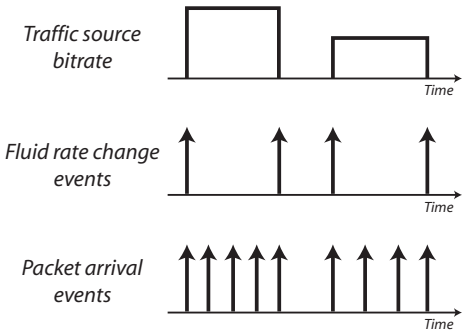


Figure 2.6: Events in fluid simulation and in packet-based simulation

much smaller event rate. The fluid flow abstraction was first used in [4] and has recently received attention from the network simulation research community [60, 67].

Rate changes of a fluid flow propagate through the network at the same speed as a packet would. For example, when a new flow enters the network it changes its own rate from zero to a given value, generating a rate change event. This event then propagates towards the flow's destination and eventually arrives at a router. If the router has enough bandwidth available to forward the new flow it will forward the rate change event unaltered. Otherwise, the new flow will have to share the available bandwidth with the other flows going through the router. Depending on the router's queuing discipline this may cause rate changes in all concerned flows, generating several new rate change events that propagate through the network and may in turn cause rate changes themselves. This is called the ripple effect. As long as the rate change events in a simulator are few, this effect does not have much impact. In large-scale network scenarios however it may become a dominant factor, effectively making the fluid simulator run slower than a packet-level simulator.

The efficiency of fluid simulation as compared to packet-based simulation, and especially the ripple effect, has been investigated in [73] and later in [72, 29]. The authors found that for a simple tandem queuing network, the number of events generated by fluid simulation grows quadratically with the number of queues while the number of events generated by a packet-based simulator only grows linearly. Conversely, fluid simulation shows sub-linear growth if we keep the number of queues constant and increase the number of contending flows. Packet-based simulation still exhibits linear growth in this scenario. The authors have also found the boundaries at which fluid simulation becomes better than packet-based simulation in the case of tandem queues. A simulation experiment in [29] using the Abilene network as topology shows that fluid simulation performs well as long as there is no congestion in the network. For the given scenario, fluid simulation reached a reduction in the number of events of up to an order of magnitude compared to packet-based simulation. However, above an average link load of 90% the fluid simulator slows down rapidly and becomes far slower than the packet-based simulator.

In general, packet-based simulation is more efficient than fluid simulation if both the number of nodes and the number of flows in a scenario are high. Furthermore, fluid simulation performs best if the application traffic follows an

on/off pattern. This is not the case in the Internet where constantly adapting protocols like TCP dominate.

The efficiency of fluid simulation can be vastly improved by abandoning the pure discrete event simulation approach. Time-stepped fluid simulation [142] splits the simulation time into short intervals during which the bitrates of the flows do not change. Because the rates of the flows are updated in discrete intervals, the frequency at which the ripple effect can occur is restricted. This simplification naturally brings about additional error, which gets bigger with increasing step size. The authors give an upper bound for the error caused by time-stepping that is proportional to the step size. When running a simulation we can thus find a suitable tradeoff between accuracy and speed. Time-stepped fluid simulation has been shown to perform well for feed-forward networks with a single traffic class. Unfortunately, it may be hard to model the dynamics of TCP using the approach.

The hybrid modeling framework presented in [11] follows a similar approach but is also able to model the behavior of TCP flows. It averages state variables over very short time scales on the order of a round trip time. Additionally, it captures the dynamics of the flows using ordinary differential equations, which allows to model TCP behavior quite faithfully. As such, it fills the gap between packet-based simulators and fluid simulators.

Another approach to combine time-stepped simulation with TCP is the Time-Stepped Hybrid Simulation (TSHS) from [42]. Like the approaches discussed above it models flows as fluid chunks in-between time steps. In contrast to these methods, fluid chunks are dissolved into a series of packet events when they reach the endpoint of the flow. The TCP model can then react to single packet arrival events like the original protocol would. Any acknowledgment packets returned by the TCP model will be collected in another fluid chunk before being sent back. Packets in a fluid chunk are assumed to be equally distributed over the chunk's time interval. For more realism the chunks may also contain a small event list for the packets they contain. Furthermore, TCP reacts strongly to bursty drops, which are commonplace with drop-tail queuing discipline in the network. A naive loss model for fluid chunks would drop single packets out of randomly chosen chunks and thus fail to model the correlations between packet losses observed in real networks. TSHS mimics this behavior using a simple algorithm: It randomly chooses a chunk and drops as many packets as necessary from this chunk. If the chunk is empty and more packets need to be dropped, the procedure repeats. TCP flows simulated with this approach show very close resemblance to TCP flows simulated using a packet-based simulator.

A concept similar to fluid chunks is used in [1]. This technique, called Flowsim, abstracts closely spaced sequences of packets of the same flow into packet trains. A threshold value serves to find the borders of packet trains. Packets are appended to a train as long as the time interval between the train's tail and the packet's arrival is smaller than the threshold. Otherwise, the packet starts a new train. See Figure 2.7 for illustration. The granularity of Flowsim can be adjusted through this threshold value. Large values result in fast but coarse-grained simulation while small values make the simulation slower but closer to packet-based simulation.

Fluid chunks hide the details of single packet arrivals in order to speed up simulation. Hao et al. [45] follow the same idea with a quite different technique. Their simulator replaces parts of an ATM networks scenario with analytical

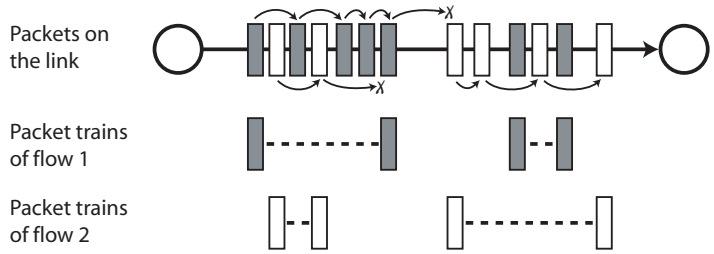


Figure 2.7: Closely spaced packets form packet trains unless their inter-arrival time is greater than a threshold

traffic models (i.e. aggregate on/off models) during a simulation run if it detects the possibility to do so. This simplification may result in speedup of one order of magnitude while maintaining accuracy of end-to-end latency within 5% of the original scenario. Unfortunately, the conditions under which this approach works are rather strict. The routes in the network must be fixed and the background traffic must be stationary, a condition rarely met by wide area networks.

It is often desirable to support fine-grained packet-based flows and coarse-grained abstracted flows in the same simulator, especially if we want to study the behavior of a few important flows that encounter background traffic in the network. Kim [61] proposes a combination of packet-level simulation for foreground flows with network-calculus-based simulation for background flows to achieve this goal. The flows of both kinds influence each other's throughput and delay. This technique results in speedup up to an order of magnitude for large-scale TCP networks while keeping the error down to 1–2% of the bottleneck bandwidth.

The Narses simulator [38] also makes simplifying assumptions about the traffic in the network. Bottlenecks are expected to occur on the first link (e.g. the modem link) of a flow or at least in the access network. The network core should be congestion free, allowing for simpler and more efficient router models. Narses uses a simple, flow-based traffic model focusing on the throughput of the flows. A flow allocates just as much bandwidth as it receives in its bottleneck link. Network delay is not considered in [38].

The abstractions presented so far in this section have one thing in common: They only abstract the flow of packets on the network but do not simplify the structure of the network itself, even though this might be beneficial for the simulation of high-level protocols. Topology abstraction has been used in [44] to simulate the Internet-scale behavior of the border gateway protocol (BGP) [105], which has seen wide-spread deployment as a wide area routing protocol. In order to attain the scalability needed for Internet scale scenarios the simulator only uses an AS-level topology and models BGP messages purely on the application layer (The abbreviation AS stands for *autonomous system*, a BGP term for a routing domain). Although this abstraction is very application specific, it allows for studying the behavior of BGP in very large-scale scenarios.

The IP multicast simulator from [51] abstracts the modeled protocols in a similar way. Instead of modeling the single hops, the multicast tree may be treated as a collection of direct source-to-destination connections. Naturally,

this is only a justifiable abstraction if there is no congestion in the network. A second optimization implemented in this simulator is to skip the flood-and-prune tree setup technique of IP multicast and use a pre-constructed tree.

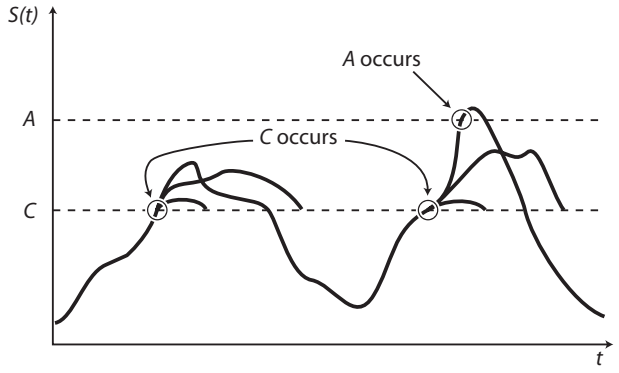


Figure 2.8: The RESTART simulation method

Most approaches that use abstraction simplify only on a few select aspects of the networks they model. The SHRiNK (Small-scale Hi-fidelity Reproduction of Network Kinetics) method however scales down the entire simulation scenario [86]. Predictions for the original scenario can then be obtained by extrapolating the results from the scaled-down version. SHRiNK reduces the number of flows in the network, the capacity of the links, and the available buffer space on routers by the same factor. For IP networks with active queue management, populated with TCP, UDP, and web flows, performance measures such as queuing delay and drop probability remain virtually the same. The simulation results become "noisier" however. Unfortunately, this promising technique fails in the case of drop-tail queuing disciplines because the scaled down version cannot mimic their bursty dropping behavior in sufficient detail. SHRiNK works best with smooth dropping behavior that can be approximated with a Poisson process.

In some situations we are not interested in the average performance of a network but rather in its behavior in extreme cases. High-reliability system for example are often advertised as having 99.999% uptime, although in order to estimate such failure rates with acceptable confidence a simulation must run very long. The RESTART (REpetitive Simulation Trials After Reaching Thresholds) [137] method uses a simple but powerful approach to estimate the rate of rare events. For a rare event A we find a more common event C so that $A \subset C$ and $p\{A\} \ll p\{C\} \ll 1$. Because of $p\{A\} = p\{C\} \cdot p\{A|C\}$ we can thus focus on two smaller problems: estimating the rather frequent event $p\{C\}$, and estimating $p\{A|C\}$. RESTART simulates a scenario like any other simulator, until C occurs. When this happens, the portion in which C occurs is repeatedly simulated. After a number of trials the simulator proceeds as normal (Figure 2.8). The advantage of this approach is that fine-grained simulation is restricted to the "interesting" parts of the scenario.

Chapter 3

Peer-to-Peer and Overlay Networks

3.1 Introduction

A major factor for the Internet's success has been its focus on end-to-end mechanisms. Protocols such as TCP do not rely on any specific functionality of the underlying network other than its ability to transmit packets from one endpoint to another. They adapt to changing conditions in the network without any explicit signaling from within the network. This transparency of the mechanisms in the network has enabled tremendous growth of the Internet without major changes of the architecture. Traditional end-to-end protocols continue to work even though new technologies on the physical layer and new routing protocols have been deployed.

The Internet's focus on end-to-end mechanisms also has drawbacks. For example, there is no generally accepted way for an application to make quality of service reservations on the network path between two endpoints. Efforts like the integrated service (IntServ) architecture have only had limited success. This may be partly due to technical reasons like the scalability of the approach or the requirement to replace many of the routers in the Internet. However, the Internet is also made up of a very large number of interconnected networks owned by independent parties. These parties have to reach a common agreement if a new technology shall be globally deployed, which is very hard to achieve. Thus, the Internet's explosive growth has also made it slow to adopt new technologies.

In recent years, peer-to-peer systems have not only become popular amongst Internet users. They have also received considerable attention from researchers because of their desirable properties, mainly their high scalability and robustness. Traditional client-server architectures employ one server to handle the requests from a large number of clients. The server machine and its Internet connection must be sufficiently fast, which can become very costly. Furthermore, if the server fails or its Internet connection breaks the service becomes unavailable. Peer-to-peer systems on the other hand are often made up of thousands of nodes distributed throughout a large part of the Internet, each taking the role of both a server and a client. This massively distributed approach reduces the performance requirements on a server to the level where it can be run

as a background process on a user's computer. Moreover, node failures only affect a small part of the service.

Another aspect of peer-to-peer systems may be even more important in the future. Peer-to-peer systems often create a logical topology between its member nodes, called an overlay network, that has no direct relation to the topology of the physical network. Since overlay topologies are purely logical they can be tailored to the needs of the applications that use them, and they can also rapidly adapt to changing conditions. Since overlay networks normally rely on a pure end-to-end approach they can operate without any support from the underlying network. This approach may be the key to adding new functionality and services to the Internet since it requires little or no changes to the physical network.

However, constructing efficient overlay topologies is not trivial. If we randomly connect overlay nodes, the logical links between them may span many physical links on the underlying network. Thus, a message sent from a node to its neighbor node may in reality travel the distance between two continents. Therefore, we need mechanisms to determine the possible cost of a logical links, depending on the nature of the overlay network. In many cases it is sufficient to know the approximate round-trip time between node pairs. Architectures like end system multicast may require knowledge about more sophisticated quality of service properties like the bottleneck bandwidth on the path between two nodes. In general, we use the term *distance* to cover all these possible quality of service measures. A naive implementation of an overlay network might measure the round-trip time between each pair of overlay nodes. This does not scale well because the number of possible links grows quadratically with the number of nodes, resulting in considerable network load just from the relatively cheap round-trip time measurements. This problem is addressed by so-called distance estimation services, that estimate the distance between overlay nodes based on a small, strategically chosen number of measurements.

There are basically two families of distance estimation approaches. One family performs direct measurements between a number of strategically chosen nodes and tries to relate these measurements to other nodes in the network based on their proximity to the measuring nodes. A prominent member of this family is IDMaps [33]. Approaches from this family often support not only round-trip time distances but also other kinds of distance measures. The other family are the so-called coordinates-based approaches. Here, every node measures its distance to a small set of nodes and tries to infer its coordinates in a metric space from the measurement results. The distance between two nodes can then be estimated using the corresponding distance function. Coordinates-based approaches generally only support round-trip time distances. GNP [82] is probably the most well-known coordinates-based approach.

In this chapter we give a brief overview of peer-to-peer and overlay networks in Section 3.2. Then, we discuss approaches to help the construction of locality-aware peer-to-peer and overlay networks in Section 3.3, and distance estimation approaches in Section 3.4.

3.2 Related Work on Peer-to-Peer and Overlay Networks

Client-server architectures are a widely-used design principle in the Internet that employs one server to handle the requests from a number of clients. For large clients populations the server machine and its network connection must be adequately dimensioned, which can become very costly. During recent years the focus of many developers has thus moved away from the standard client-server model towards more scalable and robust distributed designs. Especially peer-to-peer and overlay networks have gained in popularity.

In peer-to-peer designs the participating peers communicate directly with each other, while in client-server designs communication always goes through the server. Peers can be client and server at the same time, each having only part of the responsibility a central server would have. The computational load and network traffic created in a peer-to-peer system is consequently shared among its peers. The system is also more robust than a client-server design would be since a failing node or network link will only affect part of it. In contrast, a failed server in a client-server system affects all participants.

Because of their distributed nature one of the main challenges for peer-to-peer systems is to maintain a structure that allows peers to route requests towards a participant who can serve them. This is generally achieved through logical, application topologies connecting the peers. Such logical networks that do not have a direct relationship to the structure of the underlying network are known as *overlay networks*. The choice of topology and routing algorithm is key to scalability and robustness of peer-to-peer networks.

Peer-to-peer principles can be found in many traditional Internet protocols such as the Simple Mail Transfer Protocol [99], the Network News Transfer Protocol [59], or the Internet Relay Chat Protocol [85]. The approach has seen renewed popularity with the advent of file sharing services like Napster, Gnutella [39], and Freenet [18] even though their focus on file sharing raised several legal issues. Despite their popularity, these systems have several shortcomings. Napster relied on a central directory server and only used peer-to-peer for file transfer. In Gnutella the peers broadcast queries throughout a random topology, which entails a number of problems. For example, queries may fail to find a file even though it exists in the network. Furthermore, the broadcasted queries may accumulate, using up all available bandwidth at some peers.

Structured Peer-to-Peer Networks

A new generation of *structured* peer-to-peer networks solves the problems of the *unstructured* (i.e. randomly connected) ones like Gnutella and Freenet. Structured peer-to-peer networks choose a network topology that aims specifically at finding information in an efficient way. The central concept is key based routing. An object or a piece of information is identified by a key (usually an integer key derived from a hash value or randomly chosen). Messages are then iteratively routed towards their destination key. This method allows for applications such as distributed hash tables (DHT) and decentralized object location and routing (DOLR). The fundamental advantage of a structured approach is that each iteration brings the message closer to its destination. The random graphs of

unstructured peer-to-peer networks do not allow this, making flooding the only reliable (but inefficient) way to find information.

Chord A typical example of a structured peer-to-peer network used as a distributed hash table is Chord [129]. In a Chord network, every node (i.e. peer) receives a random ID of m-bit length out of a circular key space. The nodes form a circle in which each one only has to know the node with the next higher ID, its *successor*. Furthermore, we compute an m-bit hash value for every piece of information we would like to store (e.g. using a SHA-1 hash [121]). Key k is assigned to the first node whose identifier is greater of equal to k. Chords basic routing scheme is to follow the circular structure of the Chord network until it finds the appropriate node. However, it uses a simple trick to speed this up. Every node maintains a so-called *finger table*, which contains entries $i = 0, \ldots, m-1$ with contact information pointing to the node responsible for key $n+2^i$ (where n is the node's ID). Figure 3.1 shows an example for the effect of finger tables in a Chord network with $m = 5$. The number of hops a message takes in a Chord network of N nodes scales with O(log N).

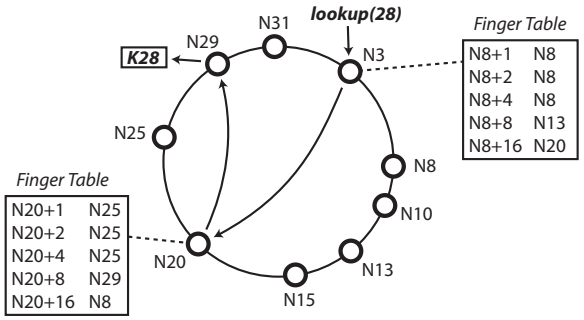


Figure 3.1: Message routing in Chord

Plaxton-based Schemes Pastry [109] and Tapestry [146] each use a variant of Plaxton routing [96] to achieve efficiency similar to Chord's. Keys in Plaxton routing consist of several b-bit digits. Nodes are assigned a random key each and build a routing table according to this key. Let us assume that node N has been assigned key 2130 with base $b = 2$. N creates a routing table with 4 rows and $2^b - 1 = 3$ entries. The first row contains links to nodes with IDs $0*$, $1*$, and $3*$ ($*$ is the wildcard character); the second row contains links to nodes with IDs $20*$, $22*$, $23*$; et cetera. Nodes will iteratively forward a message using the route with the longest matching prefix to the message's key. If the exact key cannot be found the message will end up at the node that is numerically closest to it. Figure 3.2 shows how routes are found towards key 2130. Each route is labeled with the matching wildcard. If all nodes in a network have complete routing tables, the number of hops a message takes does not exceed the number of digits in a key. In our example, a lookup for one of the 256 possible keys would take a maximum of 4 hops.

Even though Pastry and Tapestry both use a variant of Plaxton routing, they differ considerably. In addition to the routing table, Pastry nodes maintain a *leaf set* and a *neighbor set* of peer nodes. The leaf set contains nodes with

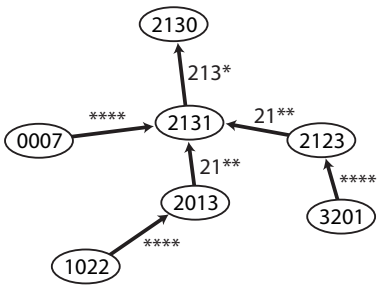


Figure 3.2: Plaxton routing paths towards 2130

numerically close IDs. Since Pastry considers the key space as circular the nodes in the leaf set are from a small sector around the current node. This set is used to make the system resilient to node failure. A node can deposit replicas of stored information on the nodes in the leaf set. Thus, when the node fails the network can be repaired without loss of information. A node's neighbor set contains nodes from its vicinity in the underlying network. New nodes use this set to refine their initial routing tables and fill them with links to nearby nodes whenever possible. This reduces the average latency of a lookup. We use Pastry as a building block for the architecture of our distance prediction service described in Chapter 5.

Tapestry employs Plaxton routing for decentralized object location and routing. A node may publish the location of an object it holds using a PublishObject message, which stores a pointer to the object at the node responsible for the object's unique key (the object's *root*). Other nodes send requests to this object using RouteToObject messages. They are first routed towards the object's key to resolve the location, and then to the object itself. Tapestry optimizes lookup by caching pointers to the object along the path from the object's location to its root. Thus, a RouteToObject may find a pointer before it reaches the object's root and will then be directly sent to the object.

Kademlia The Kademlia peer-to-peer network [77] uses a XOR metric to do a variant of key based routing also reminiscent of Plaxton's approach. The distance between Kademlia's 160-bit keys is determined by their bitwise exclusive or (XOR). Kademlia nodes keep a list of 160 k-buckets. Bucket i contains the closest peers whose IDs have a distance between 2^i and 2^{i+1} to the local node's ID. The buckets are kept up to date by monitoring the messages going through the node. Peers in a bucket may be replaced using a least-recently seen eviction policy. When a node wants to look up a key x, it selects the α nodes from its buckets that are the closest to x. Then, it asks each of these nodes for the k closest nodes to x from their buckets. From the resulting set it selects again the α nodes and repeats the process. The algorithm stops when on of the nodes returns the value for x.

CAN The key space in structured peer-to-peer network does not need to be one-dimensional. Content-Addressable Networks (CANs) [103] use a hash function that produces d-dimensional Cartesian coordinates that map to a d-torus. Nodes choose a random position inside this space and store the keys belonging

to the area around them. Each peer also keeps a list of its direct neighbors for routing. The first node to join will be responsible for the whole coordinate system. Each following node first locates the node responsible for the area in which its position lies. Then, it splits this area along one dimension and claims control over one part. Since the keys are uniformly distributed this procedure results in a quite even partitioning of the coordinate space when enough nodes have joined. Routing in this overlay network follows the logical coordinates of nodes and keys. When a node receives a message to a key x that does not belong to its own area it will forward the message to the neighbor whose coordinates are closest to x. If this neighbor has failed it tries to route around the failure by trying other neighbors in the sequence of their distance to x. The average number of hops resulting from this routing scheme is $(d/4)(n^{1/d})$, where d is the number of dimensions and n the number of peers in the system.

Peer-to-Peer Networks Providing Network Services

While the above peer-to-peer networks provide new functionality to network applications, many others focus on improving existing Internet services. Resilient Overlay Networks (RON) [2] for example connect dedicated machines in the Internet to form an overlay network capable of routing traffic around network outages and local service degradation. The topology can be changed in a matter of seconds after a detected outage. Thus, RON does not introduce new functionality but rather improves the network's capability to adapt, making it more resilient to various kinds of failures.

Although protocols for IP multicast have been known for several years, the deployment of a general multicast service in the Internet has been a long time coming. The MBone (short for "multicast backbone") [111] is a statically configured overlay network connecting the parts of the Internet that support multicast. A multicast session on the MBone may comprise physical as well as overlay links. Lately, several proposals for purely peer-to-peer-based multicast have been published. Examples include Overcast [54], Narada [50], SplitStream [15], ZIGZAG [133], and Multi+ [36]. Since it is often used for the transmission of real-time data such as video or audio streams, overlay multicast is particularly sensitive to the choice of overlay topology and potential service degradation in the network.

3.3 Peer-to-peer and Overlay Network Construction

Many systems supporting topology awareness provide distributed algorithms for creating efficient overlay topologies that are optimized with respect to the structure of the underlying physical network. Such algorithms commonly utilize a method to let overlay nodes distinguish between long, inefficient overlay links to distant peers and short, efficient ones to peers in their vicinity. This is often sufficient for significant improvements to the performance of overlay networks, because if we have the choice between sending a message to a far away node on the overlay network or to send it to a nearby node it is generally more efficient to send the message to the nearby node.

Because of the potentially large number of member nodes, a system for building efficient overlay topologies cannot consider every possible combination of logical links. The problem of constructing a truly optimal topology for an overlay network is known to be NP-hard, even if the distances between all nodes are known [37]. Clustering and grouping are frequently seen solutions to this problem. It is often sufficient for an overlay node to distinguish between peers from its local cluster and peers located somewhere else in the overlay network. In this document we make a distinction between the terms cluster and group. We use the term *cluster* for sets of nodes that are, for example, near to each other in the Internet topology. Nodes belonging to a cluster are not necessarily aware of that fact. Conversely, if the nodes are aware of the cluster they belong to and can distinguish between nodes from their cluster and other nodes from outside the cluster, we use the term *group*.

Binning A concept useful for adding topology awareness to many structured peer-to-peer networks is *binning* [104]. Nodes are assigned to *bins* based on their distance to a small set of landmarks such that a bin contains nodes from roughly the same area in the underlying network. The authors propose a scheme based on the Content-Addressable Network scheme described in Section 3.2, which maintains a d-dimensional Cartesian coordinate space partitioned into zones. While the coordinate space of the original CAN is completely logical and bears no relation to the physical topology of the network, this approach can be altered to match the above binning concept by using topology-dependent coordinates instead of purely logical ones: Each overlay node measures its distance to the landmarks and sorts the results in ascending order. For example, if a node measures 21ms round-trip time to landmark l_1, 7ms to landmark l_2 and 89ms to landmark l_3, the resulting ordering will be $l_2 l_1 l_3$. The rationale behind sorting the landmarks is that nodes with similar orderings of the landmarks are probably close to each other in the underlying topology. Now, given a set of m landmarks we can have $m!$ possible orderings. Accordingly, we divide the logical space into $m!$ bins by dividing the first axis into m partitions, the second axis into $m - 1$ partitions, and so on until we have created all possible bins. Nodes can then be assigned to a bin according to their landmark ordering. A refined scheme for binning considers not only the ordering of landmarks but also the absolute round-trip time to each of the landmarks. The round-trip times are classified into ranges, for example the ranges 0–10ms, 10–50ms, and 50ms and above. The bin in the above example would then be $l_2 l_1 l_3$: 012. A drawback of this topology-aware binning scheme is that the nodes are not evenly distributed throughout the virtual coordinate space, a problem that does not occur with a purely logical assignment to bins. The authors propose dynamic splitting and merging of bins to even out the number of nodes per bin.

The construction of peer-to-peer topology can benefit from the binning scheme because nodes from the same bin are usually close to each other and links between them are therefore short and efficient. Nevertheless, creating a topology only from short links tends to result in inefficient routing of peer-to-peer messages. A better scheme is to use a mix of short and long links. If every node has k links, a good approximation of an optimal topology can be obtained by making half of the links to nodes in its immediate neighborhood

(i.e. its bin), and half of the link to random nodes in the network. The authors call this scheme BinShort-Long and have shown that it is able to produce very efficient topologies.

mOverlay Schemes that use a fixed set of landmark nodes tend to be not very robust. If a landmark node fails it becomes impossible for joining nodes to find their place in the topology. By choosing landmarks dynamically we can get around this problem and improve robustness. *mOverlay* [145] finds groups of nodes that are close to each other in the network using a dynamic landmark scheme. The group locating algorithm of the distance prediction service we propose later in this document is partly based on this scheme. Accordingly, we discuss the mOverlay algorithm in detail.

Each group in the overlay network has a leader that keeps track of the group members and serves as contact point for nodes outside the group. It also maintains a table of the n nearest neighbor groups and the round-trip times to each of them. These neighbor groups serve as dynamic landmarks. A node joining the overlay network determines whether or not to join a group by making round-trip time measurements to the group's neighbors and evaluating the following *grouping criterion* [145]:

> When the distance between a new host Q and group A's neighbor groups is the same as the distance between group A and group A's neighbor groups, then host Q should belong to group A.

New nodes iteratively search for a group that meets this grouping criterion. When a node joins the overlay network it first contacts a *rendezvous point* (RP) and obtains contact information for a set of randomly chosen boot hosts. For each boot host it starts a locating process, which tries to find a suitable group for the node. Using several locating processes increases the robustness of the approach. The algorithm starts by contacting a boot host, which returns a message containing its neighbor table. The joining node then measures and compares its own distances to these neighbor groups. If the grouping criterion is met the process terminates and the node joins the group of the boot host. Otherwise, the new node chooses the neighbor group that is nearest to it and repeats the process. After a predefined number of unsuccessful iterations, or if all available groups have been visited, the new node creates its own group. When a node creates a new group it selects its neighbors from the closest groups it has seen during the locating process, and their neighbors. It then contacts each of the selected neighbors in order to allow them to adjust their own neighbor tables if needed. Furthermore, it must tell the rendezvous point about the new group.

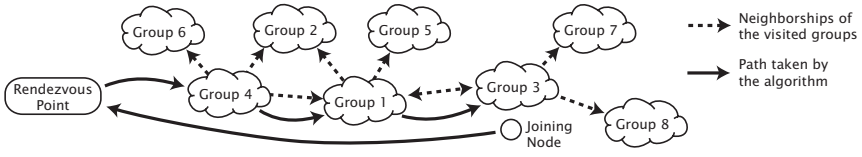


Figure 3.3: A new node joins and mOverlay network

Figure 3.3 illustrates the locating algorithm. Solid arrows indicate steps in

the algorithm and dashed ones indicate neighborships between the groups that are used to check the grouping criterion. In this example, the node starts with group 4, unsuccessfully checks the grouping criterion using groups 1, 2, and 6 as dynamic landmarks and chooses group 1 as the next hop. Here, it repeats the procedure using groups 2, 5, and 3 (the round-trip time to group 2 does not have to be measured again). The next hop, group 3, is already the best candidate for joining. Depending on the grouping criterion, the joining node now joins group 3 or it creates a new group.

Given a sufficiently robust implementation of the rendezvous point mechanism (which can be easily distributed to several servers), mOverlay's dynamic landmark procedure can be quite robust. The topology however, which is purely constructed from short links, may be prone to so-called net-splits. A net-split occurs if the joining procedure cannot reach a part of the network due to node failure. This kind of condition is less probable to occur in topologies that have both short and long links. Another potential problem of mOverlay is the group leader, a single point of failure. The authors propose to assign several backup group leaders but do not go into much detail.

Apart from its use as robust variant of a binning scheme, mOverlay has a desirable property related to distance estimation (distance estimation approaches are discussed in Section 3.4): groups identified by mOverlay can be considered as equivalence classes with respect to the round-trip time to other groups, with little error. Thus, in order to get an estimate for the round-trip time between two nodes it is sufficient to look at the round-trip time between their respective groups. This property can be used as the basis for a distance estimation service. We will focus on this topic later in this document.

Meridian In some cases, we are only interested in finding the closest node in an overlay network in relation to a given Internet host, irrespective of whether that host is a member of the overlay network or not. This problem is known as a *closest node search*. Meridian [139] is a "framework for performing network positioning without embedding nodes into a global virtual coordinate space." Apart from closest node search it is also aimed at central leader election and multi-constraint search. We base part of our group locating algorithm described in Section 5.3 on Meridian's closest node search.

Meridian nodes form a loosely connected overlay network. They exchange information about other overlay nodes using a gossip protocol and keep track of a fixed number of peer nodes. These nodes are sorted into non-overlapping, concentric rings of exponentially growing width around the Meridian node (see Figure 3.4). The ith ring contains nodes with latencies between αs^{i-1} and αs^{i} from the center, and the outermost ring contains nodes with latencies αs^{i^*} and more. Within each ring, the nodes are selected to maximize diversity, which is quantified through the hyper-volume of the k-polytope formed by the selected nodes. A closest node search aims to identify the Meridian node that is closest to a given end system E in the network. To start the procedure we send a request to an arbitrary Meridian node. This node measures its latency to E and selects the nodes with similar latencies from its cache. It then contacts each of these nodes and asks them to measure and report their latency to E. The node with the smallest latency to E becomes the next hop, and the procedure repeats. When the next hop is only insignificantly closer than the current one the closest node

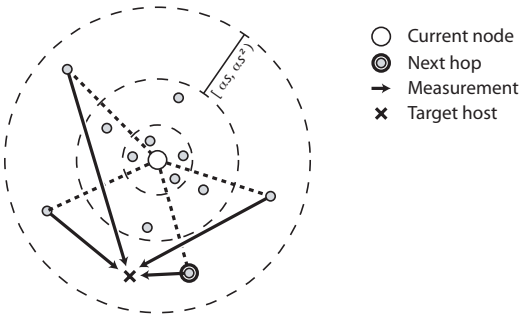


Figure 3.4: A step in Meridian's closest node search algorithm

search terminates and the current node is selected. Central leader election is an extension to this problem where we have several end systems E_i $(i = 1, \ldots, n)$ and search for the overlay node with the smallest average latency to all these end systems. The above algorithm can easily be adapted to solve this problem: Each time node would measure the latency to E it measures the average latency for all E_i instead. The rest of the algorithm remains the same. A multi-constraint search finds overlay nodes located at given maximum distances d_i from end systems E_i. In this variant of the algorithm, the current node chooses any nodes from its rings as next hops whose distance to the current node does not imply that it cannot satisfy the constraints. For example, if the current node's distance to E_i is 30ms and the constraint is 10ms, a node 5ms away from the current node would not be chosen as next hop.

Meridian has several desirable features. Its loose structure makes it resilient to node failures. Furthermore, because of it's light-weight design it can be deployed in networked computers without having a big impact on their performance.

Netvigator Sharma et al. present a system called Netvigator [122] that is both an application and a refinement of the landmarks-based approach. Its main goal, like Meridian, is to find the closest server to a client node from a set of candidate servers . For this purpose it employs a landmark-based approach with an extension called *milestones*. Milestones are intermediate routers between the client node and a landmark that do not only forward the probing packet but also send a reply back to the client node. In this way they can be dynamically discovered while the client node measures its distance to the landmarks. When the client node has finished measuring it sends the resulting distance vector (including the distances to the discovered milestones) to a central repository. There, a clustering algorithm is employed to find the k candidates closest to the client node and returns the set. The client node now uses direct measurements to find the closest server from this set. Even though Meridian is very similar in nature, the two approaches have to the best of our knowledge never been directly compared. Nevertheless, we can say that Netvigator would be comparatively complex to deploy because it requires changes to the network infrastructure to support the notion of milestones.

3.4 Distance Estimation

A useful class of systems supporting topology awareness are those concerned with estimating distances between Internet endpoints. These systems allow clients to query the distance between two endpoints without having to measure the distance themselves. In this context, network distance is often a synonym for round-trip time, but it may also encompass available bandwidth between peer-to-peer nodes or delay jitter on the path between them. For large systems it is rarely feasible to actively measure the distance between each pair of nodes because of the significant network load even simple measurements like ICMP ECHO messages may entail. The amount of such measurements grows quadratically with the number of nodes in the system. This scalability problem is particularly emphasized in distance estimation approaches that consider the whole of the Internet and not only the nodes belonging to a specific peer-to-peer or overlay network. The greatest challenge of network distance estimation consequently is to infer good estimates for end-to-end distances from only a small number of actual measurements. We divide end-to-end distance estimation approaches into several subclasses. The first subclass uses direct measurements between a restricted set of Internet nodes and aims to map or combine these measurements so they can serve as distance estimates for pairs of nodes that do not make measurements themselves. Since they generally cluster nodes for this purpose we call them clustering-based approaches. In the second subclass of distance estimation approaches each node makes a small set of measurements to special nodes in the Internet and computes synthetic coordinates from these. The distance between two nodes can then be estimated by applying a distance function to their respective coordinates.

3.4.1 Aspects of Distance Estimation Approaches

The many proposed variants of distance estimation all have advantages and disadvantages. In the following we will discuss what we believe are the most important aspects of distance estimation services.

Scope – Many distance estimation services can only provide estimates for the distance between pairs of nodes that actively participate in the system. Coordinates-based approaches for example require each node to compute its own coordinates, making it impossible to estimate distances to uninvolved nodes. Even though they are designed for global deployment, these architectures can only succeed if a positioning software is running on every end system. This "member scope" is sufficient for many applications. For example, a file sharing peer-to-peer network does not need any distance information about nodes that do not participate in the network since all communication takes place between members of the peer-to-peer network.

Nevertheless, a "global scope" distance estimation service for any pair of Internet nodes would be useful to a much wider range of applications not necessarily restricted to peer-to-peer and overlay networks. Several approaches are aimed at providing distance estimates for any two reachable IP addresses in the Internet. This is achieved by dividing Internet nodes into clusters and relating the distances between these clusters to the distances between overlay nodes.

Infrastructure requirements – The success of many peer-to-peer networks is partly due to their ability to run solely on end user systems, without any dedicated servers or additional infrastructure in the network. As for every service, the cost and complexity of deployment is an important factor for its acceptance. Some approaches rely heavily on deploying special servers at strategic points in the Internet. The more servers are deployed and the better their placement, the better the resulting distance estimates. Distance estimation systems using a more distributed approach have a clear advantage in this respect. For example, some coordinates-based approaches calculate the nodes' coordinates in a totally distributed way and require no additional infrastructure at all.

Required deployment size – Every distance estimation approach requires a minimal number of nodes placed at a sufficient number of different locations to be able to yield acceptably accurate results. In some fully distributed approaches the number and distribution of the nodes automatically grows with the size of the network. Conversely, approaches that rely on a fixed set of servers depend heavily on the servers' placement. If they should adequately cover the whole of the Internet we would have to deploy servers in various places around the globe.

Scalability and robustness – A key consideration for any distance estimation scheme is its scalability and robustness. Especially the approaches aiming for global deployment need to handle a very large number of client requests. Moreover, the failure of a single node should not render the service unusable. Systems using peer-to-peer designs often have an advantage in this respect.

Available types of distance estimates – Most distance estimation services only provide estimates of the round-trip time between hosts. Round-trip time is a popular choice because it is very cheap to measure. However, there are many other useful possibilities, like available bandwidth and delay jitter. Services that can provide several types of distance estimates are useful in more application scenarios than those concentrating on round-trip times.

Prediction vs. estimation – Network conditions in the Internet change constantly. Human activity produces daily and weekly cycles of network load, and changes to the network topology and routing may alter an end-to-end path's properties permanently. Most distance estimation approaches base their algorithms on a necessarily static perception of the network, which may be adequate for the Internet core but is an oversimplification when considering end-to-end distances between nodes on the network edge. This problem is often side-stepped by using more static measures such as the minimum instead of the average round-trip time. Nevertheless, the Internet's constant changes require periodic updates of the system, and even then the estimates may be often out of date. We believe that instead of estimating static distance values we should focus on the trends of the distances. Thus, instead of the average round-trip time between A and B as observed in the past, the service should provide a prediction of round-trip time between A and B for the near future.

3.4.2 Clustering-based Distance Estimation

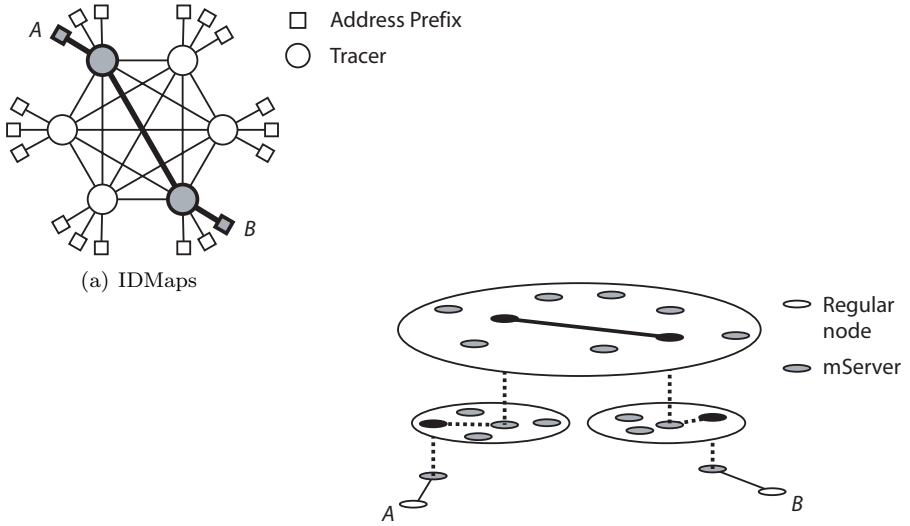
The need for a general distance estimation service in the Internet has been recognized relatively early. SONAR [80], a protocol for distance queries, and

Host Proximity Service (HOPS) [32] were discussed in the IETF as early as 1996 and 1997, respectively [33].

IDMaps The Internet Distance Map service (IDMaps) [33] is intended to be the underlying service that provides the distance estimation for SONAR/HOPS. It addresses the scalability problem of Internet-wise distance estimation using two concepts, address prefixes and tracers. *Address prefixes* are clusters of endpoints sharing a common prefix in their IP addresses. When IDMaps estimates the distance between two endpoints it really estimates the distance between their respective address prefixes. The assumption behind this is that the longer the matching prefixes of two IP addresses, the closer the corresponding endpoints. This clustering reduces the complexity of the distance estimation problem but it is not enough to make periodic measurements between all address prefix pairs tractable. However, a second level of clustering based on so-called *Tracers* improves the scalability further. Tracers are special nodes deployed around the Internet so that each address prefix is relatively close to at least one tracer. They periodically measure the round-trip time to all other tracers and to address prefixes close to them. IDMaps estimates the distance between two address prefixes as the sum of the distance from each address prefix to its nearest tracer, and the distance between the two tracers. Figure 3.5(a) shows an example for two address prefixes A and B. Naturally, the accuracy of the estimate heavily depends on the number and placement of the tracers. The choice of address prefixes is also a factor. Mechanisms like network address translation may hide a whole subnetwork behind a single public IP address, rendering the assumption behind address prefixes invalid. Considerable overestimation may result if we query the distance between two endpoints from adjacent address prefixes that use the same tracer, because the distance to the tracer and back may be much longer than the direct distance between the address prefixes.

Shavitt et al. [123] refine the relatively simple distance estimation of IDMaps. By using `traceroute` to measure the distance between tracers we also learn about the route between them. If we can find "Crossing points," common nodes on two distinct routes between tracers, we can often use methods from linear algebra to calculate the distances between crossing point and tracers. Thus, we refine the tracer topology without actually making more measurements or deploying more tracers.

Dynamic Distance Maps A critical factor for the performance and accuracy of IDMaps is the number of tracers. Too few tracers make the approach inaccurate, but too many overly increase the amount of periodic measurements. The Dynamic Distance Maps method [132] uses mServers, a variant of IDMaps' tracers that are organized into hierarchical groups. Groups that are high in the hierarchy represent large areas in the network topology while groups lower in the hierarchy give a more details picture of smaller areas. Unlike tracers, mServers only make distance measurements to other mServers in the same group. If we want to find out the distance between two mServers m_1 and m_2 that are not in the same group, we have to find a higher level group containing mServers p_1 and p_2 that are parent nodes for m_1 and m_2, respectively. The measured distance between p_1 and p_2 then serves as an estimate for the distance between m_1 and m_2. For two regular Internet hosts, the distance is estimated as the dis-



(a) IDMaps

(b) Dynamic Distance Maps

Figure 3.5: Distance estimation using dedicated infrastructure in the Internet

tance between their nearest mServers. Figure 3.5(b) shows an example where the measured distance from the second hierarchical level is used to estimate the distance between A and B.

M-coop The M-coop system [128] follows an approach similar in spirit to IDMaps but takes advantage of the BGP autonomous system level (AS-level) topology of the Internet. Each node in the M-coop overlay network has a so-called area of responsibility, defining the set of IP addresses for which it can answer queries. This area may comprise several autonomous systems, or only a part of an autonomous system depending on the number of nodes in the area. The logical topology between these overlay nodes follows the AS-level topology of the underlying network. Consequently, in order to estimate the distance between two overlay nodes we can just follow the shortest path between the nodes in the overlay topology and sum up the distances of the single hops. Because the shortest path on the topology closely resembles the path a normal packet would take on the underlying network this sum is a good estimate of the total distance between the nodes. Unfortunately, this approach requires knowledge of the current AS-level graph, and if possible the BGP routing policies of each autonomous system. The M-coop overlay can therefore only be deployed on nodes controlled by the ISP since other Internet hosts do not normally receive BGP messages. Like Dynamic Distance Maps, M-coop was designed to support more than just one type of distance measure. Measurements may be obtained actively, or passively by monitoring application traffic. Requests may trigger additional active measurements.

Advantages and Disadvantages of Clustering-based Approaches

The presented clustering-based distance estimation services all have a global scope, i.e. they can estimate the distance between any pair of reachable IP addresses. The main drawback of these approaches is their reliance on new infrastructure in the network. In order to make an accurate distance estimation between two endpoints they require measurement servers placed close to each. Because of their global scope, this may add up to a large number of servers and thus to high deployment costs. Their using fixed servers can also be a problem for the robustness and scalability of these approaches.

In contrast, our distance prediction service (described in Chapter 5) can only provide estimates if one of the hosts in question is a member of the service. However, it does not require any additional infrastructure in the network and requires only a small deployment to be useful. Its peer-to-peer structure also makes it more robust and scalable.

IDMaps and M-coop both cluster hosts according to their IP addresses and assume that hosts with similar addresses have approximately the same distance to a given measurement server. This may be an over-simplification because a significant part of a packet's route may be hidden by mechanisms like network address translation (NAT) and Mobile IP. Furthermore, finding a good partitioning of IP addresses into clusters is not easy without detailed knowledge of the network topology. Like Dynamic Distance Maps, our approach consequently relies on measurement-based clustering. However, in order to reduce the required deployment size of our service we use a clustering algorithm that is able to detect clusters based on measurements made only from a single site.

A commonality between Dynamic Distance Maps, M-coop, and our approach is that they support arbitrary distance measures. Moreover, M-coop's measurement strategy resembles the one we use. Nevertheless, both Dynamic Distance Maps and M-coop are restricted to estimating the current distance between two hosts. Our approach exploits the observable statistical properties of the measurements to make predictions of the distance in the near future and can also indicate the expected error of prediction.

3.4.3 Coordinates-based Distance Estimation

A method for Internet distance estimation that has recently received remarkable attention is to assign synthetic coordinates to network hosts. The advantage of this approach is that the distance between two coordinates can usually be calculated using a simple distance function. The coordinates used by distance estimation approaches usually have no direct relation to the measured distances. These approaches usually aim to find a set of coordinates in a metric space so that the metric distance between two coordinates is a good estimate of the network distance between the two Internet hosts they represent. Many methods use n-dimensional Euclidean space for this mapping. In general, this mapping cannot be done without introducing some error. The main challenge for these approaches is thus to find a set of coordinates that minimizes the error of embedding.

Coordinates consume very little memory. If we need to store the distances between a large set of nodes we can either store large distance matrix growing quadratically with the number of nodes, or we can simply keep a list of the nodes'

coordinates, which grows linearly with the number of nodes. Moreover, because of their small size, coordinates can be easily included in protocol messages like those sent through a peer-to-peer network. When another peer-to-peer node indirectly receives the message it can estimate its distance to the sender without any further communication or measurement.

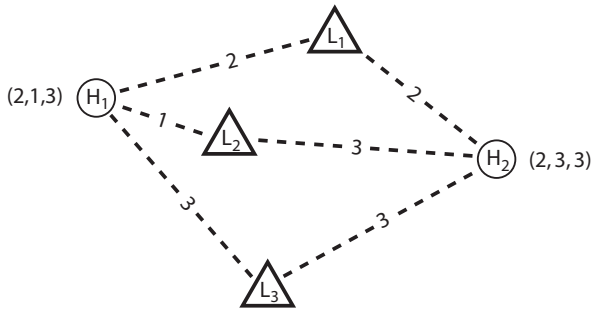


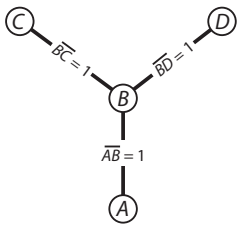
Figure 3.6: Coordinates used in the triangulated heuristic

Basic Methods

Triangulated Heuristic A basic distance estimation approach using coordinates, called *triangulated heuristic*, was defined in [49]. It was originally aimed at optimizing shortest-path searches in inter-domain graphs. In this approach a fixed set of Internet hosts take the role of landmarks. We construct coordinates for a host by making a round-trip time measurement to each of those landmarks and combining the result to form a vector (see Figure 3.6). By comparing the vectors of two hosts we can give lower and upper bounds for the distance between them. Given landmarks $L_1, \ldots, L_N$ and hosts H_1 and H_2, the lower bound for the distance $d(H_1, H_2)$ between the hosts is $L = \max_{i \in 1,\ldots,N} |d(H_1, B_i) - d(H_2, B_i)|$. The upper bound is given by $U = \min_{i \in 1,\ldots,N} \big(d(H_1, B_i) + d(H_2, B_i)\big)$. A commonly used estimate for the distance between the hosts is the average $(L + U)/2$. However, in other work [82] it has been noted that U is often the better estimate than both, L and $(L + U)/2$.

Embedding The majority of coordinates-based distance estimation approaches aim at embedding node distances in a metric space such that the metric distance between two coordinates is a good estimate of the network distance between the two Internet hosts they represent. n-dimensional Euclidean spaces are a particularly popular choice in this area. Even though the embedding of network distances in Euclidean space generally cannot be done without error the resulting distance estimates results are often surprisingly good.

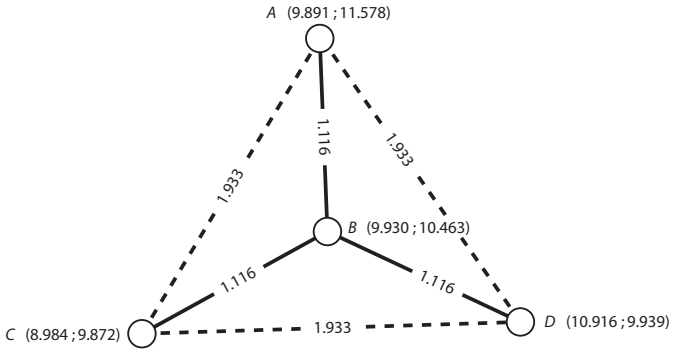
We take a closer look at the Euclidean-coordinate-based approach considering the topology from Figure 3.7(a) as an example. Four nodes connected by unit length links form a "Y" topology. The distance of two nodes is given by the sum of the link lengths between them. I.e. if two nodes are neighbors their distance is 1, otherwise it is 2. Figure 3.7(b) shows the corresponding distance matrix. A perfect mapping of these nodes to coordinates in an n-dimensional



(a) Original topology

	A	B	C	D
A	0	1	2	2
B	1	0	1	1
C	2	1	0	2
D	2	1	2	0

(b) Distance matrix



(c) Topology embedded in a 2D Euclidean coordinate system

Figure 3.7: Embedding of a simple network topology into 2 dimensional Euclidean space

Euclidean space requires that the Euclidean distances between the coordinates equal the distances between the nodes in the topology. Unfortunately, this goal cannot be achieved. In fact, no "Y" topology with link lengths greater than zero can be mapped to Euclidean space without error (see Appendix A.1 for proof). Nevertheless, we can find an Euclidean embedding that *minimizes* the error. This is the main challenge for all distance estimation approaches based on metric spaces. For our example, we can use the two-dimensional Euclidean embedding shown in Figure 3.7(c), which was computed using the GNP approach discussed later in this section. It minimizes the total square error of embedding

$$\varepsilon \quad = \quad \sum_{n,m \in \{A,B,C,D\}} (\hat{d}(n,m) - d(n,m))^2 \quad .$$

We can see that the embedding slightly overestimates the distances between neighbor nodes, and underestimates the distances between the other node pairs. $\hat{d}(n,m)$ is the estimated distance and $d(n,m)$ the actual distance between nodes n and m. The embedding from Figure 3.7(c) has a total square error $\varepsilon = .053835$. Each distance will be predicted with a relative error of approximately 7.5%. The coordinates in the example demonstrate another property often encountered in Euclidean embedding: Minimizing the error may result in any isometric mapping of the coordinate system, i.e. the system may be arbitrarily rotated and translated depending on small factors like the ordering of the landmarks during the computation.

Existing Approaches

Global Network Positioning Euclidean embedding for network distance estimation has been used for the first time in [82], in which Ng and Zhang proposed a system called Global Network Positioning (GNP). It is based on a small set of landmark nodes. GNP works in two phases: it first measures the distance between each pair of landmark nodes and computes an Euclidean embedding with minimal total square error. The minimization problem is solved with the Simplex Downhill algorithm [81] because this algorithm is able to minimize arbitrary functions without requiring additional information like the function's derivatives. Once the landmark coordinates have been computed phase two starts. A node that joins the system will receive a list of the landmarks and their precomputed coordinates. It will then measure its distance to each of the landmarks and use the resulting distance vector together with the landmark coordinates to compute its own coordinates. Like the initial computation this is done with the Simplex Downhill algorithm. However, the complexity of the problem is considerably smaller because the landmark coordinates are already known. GNP has been shown to perform quite well when applied to several distance matrices of round-trip time based on measurements from the Internet. Nonetheless, using a fixed set of landmarks makes the approach vulnerable to node failure or malfunction. If a landmark node fails, or if it maliciously distorts the round-trip measurements, the computed coordinates will be wrong and result in bad distance estimates. The network load close to the landmark nodes may also be a problem if a very large number of nodes measure their distance to a small set of landmarks.

Network Positioning System In [83], the authors of GNP present a Network Positioning System (NPS) that addresses the shortcomings mentioned above. The minimization problem for the initial computation of landmark coordinates has been reformulated to allow for decentralized computation. Furthermore, NPS introduces a hierarchical system of reference nodes. Only a small set of nodes compute their position based on the actual landmark nodes. They form the first layer. Nodes from the second layer compute their positions based on reference nodes from the first layer and serve in turn as reference nodes for the third layer. Obviously, this scheme scales much better than GNP, and is more robust to temporary failures of the landmarks, although it may also result in accumulated error in the outer layers. The authors believe that a three-layer system is the optimum for an Internet-wide service. NPS nodes also recompute their position regularly in order to adapt to topology changes.

Lighthouses An alternate approach addressing the scalability issues of GNP, presented in [94], uses dynamically chosen landmarks called Lighthouses. When a new node joins the system it arbitrarily chooses a number of local reference points from the set of nodes that have already been positioned. Then, it computes a *local* coordinate system based on these reference points and calculates its own position in it. Additionally, it computes a transition matrix between its local coordinate system and the global one. Evaluation of this distributed approach based on distance matrices of round-trip time indicate that its accuracy is comparable to GNP.

Practical Internet Coordinates Practical Internet Coordinates (PIC) [22] addresses GNP's problem of having fixed landmarks with a simpler approach. Joining nodes select landmarks from the set of nodes already in the system and compute their own coordinates using Simplex Downhill, without the additional local coordinate system used in Lighthouses. PIC nodes may select landmark by following one of two criteria: (i) They pick the n nodes closest to themselves in the network topology, or (ii) they pick some of the closest nodes and augment the set with randomly chosen nodes from anywhere in the system. The second option is reported to yield better embeddings in general. PIC also addresses the problem of forged coordinates, which may be more of an issue than in GNP because landmarks are chosen randomly whereas GNP uses a small set of easily controllable landmarks. When a node selects its landmarks, it also checks the triangle inequation on the resulting system. Violations of the triangle inequation are commonplace network distance matrices. However, very bad violations may be a sign of forged coordinates. PIC thus removes the nodes from a landmarks set that violate the triangle inequality "the most."

Big-Bang Simulation The accuracy of the distance estimates from an Euclidean embedding heavily depends on the algorithm applied to minimize the error. The Simplex Downhill method employed by GNP is simple to use because it can be applied to arbitrary functions, but other algorithms may give better results. Big-Bang Simulation [124] is such an algorithm. It models network nodes as particles traveling in the Euclidean space under the effect of potential force fields. Initially, all nodes are placed at the origin of the coordinate system. In each iteration the particles move according to Newton mechanics. They attract or repulse each other depending on whether their distance is too small or too large, respectively. The field force is derived from the total embedding error. Eventually, the total embedding error will reach a minimum and the algorithm ends. Big-Bang Simulation was compared to several other embedding algorithms including Simplex Downhill and has proven to be produce superior results in most scenarios.

Vivaldi Even though Big-Bang simulation finds very good Euclidean embeddings, it shares a shortcoming with GNP: Neither algorithm can be distributed. This may cause scalability issues when the algorithm is a applied to a very large system. A possible solution to this problem is Vivaldi [26], a fully distributed network coordinate system based on an algorithm similar to Big-Bang Simulation. Vivaldi places a virtual spring between pairs of nodes with a rest length set to the known network distance between the nodes and a current length set to the distance between the nodes in the current embedding. Depending on the difference between the current length and the rest length each spring exerts a force on the connected nodes. Like in Big-Bang Simulation the nodes are moved through the Euclidean space until they settle in an equilibrium. However, the algorithm employed by Vivaldi does not rely on a global error to detect convergence because that would make distributed computation impossible. Instead, it uses a locally calculated timestep δ, which it adapts depending on the local error observed by a node. If the error is large, δ should have large values in order to make the node move quickly to an approximately correct position. Smaller values of δ allow it to refine its position when it is close to its final

position. Making the movement of nodes dependent on their local error allows for fully distributed computation of the Euclidean embedding, although one problem remains. If we would place a spring between each pair of nodes, each node would have to constantly communicate with all other nodes, causing immense communication overhead. Vivaldi nodes consequently maintain only a restricted set of springs and update their position if they detect significant error when communicating with other nodes. A peculiarity of Vivaldi introduced in later versions is its coordinate system. The authors found that a large part of the error in Euclidean embeddings is due to the access links of the nodes. Nodes in the network core can normally be embedded with very little error. Vivaldi therefore uses 2-dimensional Euclidean coordinates with "heights" to model the access link. The distance between two nodes is thus given by their Euclidean distance plus each of their height values (see Figure 3.8). Vivaldi's greatest strength is without doubt its fully distributed structure. Nevertheless, the resulting embeddings are approximately as accurate as those found with GNP.

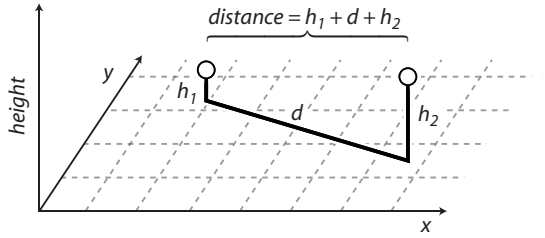


Figure 3.8: Vivaldi uses 2-dimensional Euclidean coordinates augmented with heights

Network distance estimation methods based on Euclidean embedding perform surprisingly well in simulation studies. Ledlie et al. criticize that these simulations generally use static distance matrices and argue that the highly variable round-trip time measurements on the Internet may severely impair the accuracy of the resulting distance estimates [68]. The authors propose to apply a moving-percentile filter to round-trip time measurements to make them more stable and report significant improvements resulting from this.

PCA-based Coordinates Lim et al. describe an interesting non-Euclidean variant of the coordinates-based approach [70]. They apply Principal Component Analysis (PCA) to a distance matrix of beacon nodes (i.e. landmarks) to construct an n-dimensional space with a new basis. The number of dimensions n is determined by the number of principal components whose contribution to the total variation is more than a predefined threshold. An attractive property of this approach is that calculating coordinates of joining nodes is very cheap. They simply create a vector of their distances to each beacon node (similar to the triangulated heuristic approach shown in Figure 3.6) and multiply it with the system's transformation matrix to receive their final coordinates. In contrast, the minimization algorithms used in approaches such as GNP are rather expensive.

CHESS In [75], Malli et al. argue against the common assumption that short round-trip times between two hosts on the Internet usually coincide with other desirable quality of service properties, such as available bandwidth. They present large-scale measurements showing that this assumption may lead to results that are far from optimal. Accordingly, the authors propose a coordinate-based distance estimation approach that takes application-dependent utility functions into account. The network paths between nodes that are close in this coordinate-space have high utility to the application. Depending on the application this may mean a combination of high available bandwidth and low round-trip time, for example.

Advantages and Disadvantages of Coordinates-based Approaches

The biggest advantage of coordinates-based approaches is that they can be used for more than just distance estimation. A good example is the use of coordinates for binning in content-addressable networks as described in Section 3.3.

For pure distance estimation however they have a number of drawbacks. Coordinate-based approaches are generally restricted a member scope, i.e. they can only estimate distances between hosts that have installed special software and have computed their coordinates. Approaches like GNP could be modified so that the landmark nodes actively measure their distance to a given endpoint and compute its coordinates, but this approach could hardly scale. Conversely, coordinates-based approaches require little or no infrastructure. GNP requires a small, fixed set of landmark nodes. Fully distributed systems like Vivaldi do not require any additional infrastructure at all. The increased distribution also results in better scalability and robustness.

The required deployment size of coordinates-based systems depends on the distribution of their member nodes in the network. Widely distributed member nodes also require widely distributed static or dynamic landmark nodes to compute their coordinates. Therefore, if we want to estimate distances between nodes all over the globe we also need globally distributed coordinates-based system.

Another general drawback of coordinate-based systems is that they are restricted to a single distance measure, which is usually round-trip time. Furthermore, they cannot provide information about the variance, trend, or expected error of a distance estimate.

Chapter 4

Hybrid Simulation

4.1 Introduction

In traditional, sequential packet-based simulators the simulation scenario is modeled in terms of nodes and links with individual capacities and delay characteristics. Every time a node sends or receives a packet an event is scheduled. When simulating networks at the scale of the Internet (or even parts of it) this approach quickly becomes problematic due to the sheer amount of events to be processed. Optimizing the simulator's code may result in minor speedups, but this normally isn't enough to be able to simulate large-scale scenarios in reasonable time. Sequential simulation research has been working on this problem for decades. The problem can be approached in two ways: We can make the simulation parallelizable in order to benefit from the computational power of modern high performance computers, or we can simplify the simulation model in a way that preserves the details we are interested in but only provides coarse grained simulation of other aspects of the scenario.

Parallelization (see Section 2.2.2) may result in speedups of several orders of magnitude, but this depends heavily on a good partitioning of the original network scenario. If we partition the scenario at the wrong places the communication overhead between the processors may negate the speedup gained from parallelization. Optimal partitioning is also very dependent on the parallelization approach and the simulated scenario. A partitioning that works well in one case does not necessarily work well in other cases. Setting up or changing a simulation scenario for a parallel simulator can thus be tedious. Especially in network research where we often change scenarios to explore different aspects of a new networking technology this may be a hurdle. Another problem that occurs with packet-based parallel simulations of large-scale Internet scenarios ironically stems from what is actually an advantage of the approach: it's high level of detail. Even for relatively predictable networks such as telephony networks it is hard to make good assumptions about the traffic patterns. In order to create realistic Internet scenarios we would have to consider many different protocols that often interact in unpredictable ways. Moreover, the exact topology and configuration of the network is rarely known, especially if we simulate scenarios spanning the networks of several internet service providers (ISPs). Even in known topologies we may be missing significant details like the queuing

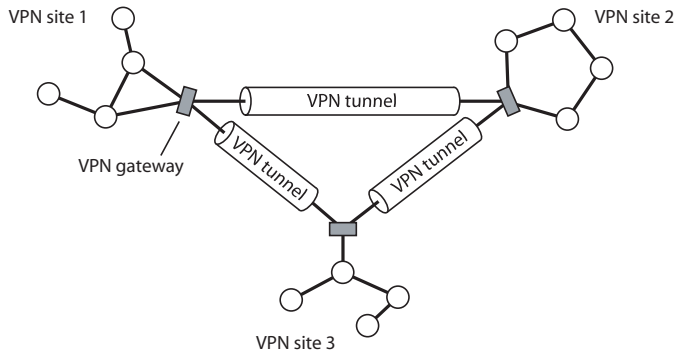


Figure 4.1: Scenario of a multi-site VPN network

disciplines used by the routers. For these reasons parallel simulation may be ill-suited for large-scale Internet scenarios even if the attainable speedup may be immense.

The second approach to solving the scalability problem of sequential packet-based simulation is to reduce the complexity of the simulation by hiding unnecessary details. Such abstractions only preserve details of interest and simplify the simulation of other aspects. It is therefore essential to choose the right abstraction with regard to the network characteristics we are interested in. In Section 2.2.3 we have presented several different abstraction schemes aimed at different aspects of the simulated network. A big advantage of abstraction-based simulators over parallel simulators is their ability to cope with incomplete or coarse-grained scenarios. For example, if our scenario includes an ISP network with unknown topology we can still make a coarse-grained model for that network based on the properties that are observable from the outside. Thus, we simulate the ISP network faithful to the available information. In a packet-based scenario however, we would have to make a guess at the inner structure of that ISP network, hoping to faithfully mimic its behavior. This is generally far from trivial.

Specific simulation approaches and their abstractions are usually implemented in their own simulator programs. This makes them only usable for the type of network or scenario they were designed for. Nevertheless, many scenarios of interest could be better simulated if we could combine different abstractions in the same simulator. Consider the example of a multi-site virtual private network (VPN) as shown in Figure 4.1, consisting of several local networks connected through so-called tunnels. When simulating a distributed application deployed at the three sites in this scenario we may want to employ a fine-grained abstraction such as a normal packet-based approach for the local networks, while using a more coarse-grained abstraction for the VPN tunnels. This is especially useful if we do not have detailed information about he network between the VPN sites and the traffic flowing through it. We may be restricted to creating a model based on end-to-end measurements between the VPN sites. This combination is not possible in most available simulators.

We have designed and implemented a hybrid simulator that combines packet-based simulation with an abstract network model for coarse-grained but efficient

simulation of multi-domain networks. It uses concepts from queuing theory and fluid simulation (see Chapter 2 for a discussion of these approaches).

The hybrid simulator is based on an extension to the packet-based simulator ns-2 that is able to introduce arbitrary abstractions into a simulation scenario. The basic idea is to extend the simulator's node model using dynamically loadable modules. When a packet reaches a node it will normally get routed and forwarded on the next link. In contrast, if it reaches an extended node the extension module will be given a chance to delay or drop the packet according to its own abstraction. Because this approach combines several different abstractions in a single simulator we call it *hybrid simulation*. The possible applications of this extension are numerous. For example, it would be possible to implement detailed router models to make the forwarding delay of the simulator more realistic. More importantly, an extension module can simulate a whole sub-network using an arbitrary abstraction. We use this concept to combine the packet-based models of ns-2 with our abstract network model for multi-domain networks. In the above VPN example, we could thus simulate the single sites using a normal packet-based approach but model the network between the sites using our coarse-grained abstraction. Figure 4.2 illustrates this concept. The extended node in the middle simulates a small network instead of just forwarding packets to the next link.

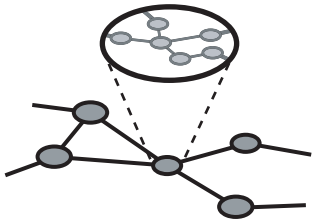


Figure 4.2: Extended nodes can simulate whole sub-networks

This approach is also particularly suitable for simulating overlay networks that require additional network infrastructure, such as the clustering-based distance estimation systems presented in Section 3.4.2. Pure packet-based simulation of these systems is hardly feasible due to the wide distribution of the measurement servers and the complexity of the network between them. Conversely, if we only model the vicinity of the measurement servers with packet-based simulation and simulate the parts of the Internet between them as multi-domain models, we obtain a very scalable simulation scenario.

We will present the multi-domain network model in Section 4.2. The extension mechanism necessary to integrate it into ns-2 will be discussed in Section 4.3.

The hybrid simulator was originally developed in the course of the Intermon project [52], which aimed at creating an integrated architecture for the monitoring, modeling (simulation) and visualization of inter-domain networks. One of the key features of this architecture is the seamless integration of monitoring, modeling & simulation, and visualization tools. For example, it is possible to schedule a BGP topology detection task, then use the detected topology to create simulation scenario, send it to a simulation server, and then analyze the

results using various visualization tools, all from the same graphical user interface. The hybrid simulator is one of several simulation tools integrated in this architecture. Section 4.4 focuses on the details of the hybrid simulator's integration, such as the generic simulation job descriptions and the automated scheduling of these jobs on a simulation server.

The work in this chapter was also presented in [8, 9, 10, 112, 114, 113] and in technical reports [115, 116, 119, 120, 66].

4.2 An Abstract Network Model for Inter-Domain Scenarios

Scalability in network simulation is often achieved by reducing the level of detail of the simulation scenario or of the simulation algorithm. Carefully chosen, such abstractions of the simulated network can significantly reduce the complexity of large-scale simulations. In this section we present an abstract network model that is mainly aimed at simulating inter-domain networks more efficiently than the traditional packet-based approach while still giving a good approximation of real network behavior. Its implementation is based on the an extension module for the ns-2 simulator described in Section 4.3.

The network model is based on the assumption that, over certain time spans, networks like the Internet can be divided into areas where congestion is negligible, interconnected by bottleneck links. This assumption maps rather well to networks controlled by a single ISP. In order to satisfy service agreements with customers, ISPs have an interest to keep their internal networks free of congestion and preferably drop excess traffic at the border links of their domains. If they didn't, the excess traffic might congest links inside the ISP's network and cause service agreements to be broken. A possible partitioning for inter-domain networks is thus to model the border links between ISP domains as potentially congested and the ISP domains themselves as congestion-free. Figure 4.3 shows this modeling view.

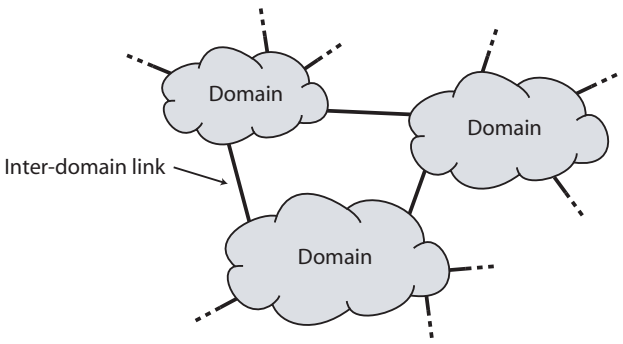


Figure 4.3: The basic modeling view

Domain Models We treat congestion free areas as black boxes, which we call *domain models*. Separate models for congestion free areas have the advantage that we can neglect packet losses and excessive queuing in these parts of the

network and restrict the model to quasi-stationary delay behavior. Apart from its scalability advantage this approach is primarily useful to model network areas of which we do not know the exact topology. Again, ISP networks are good examples since ISPs are naturally reluctant to reveal details about their inner networks for competitive reasons. The delay behavior of domain models is based on empirical cumulative distribution functions (ECDFs) simulating the stationary delays of packets crossing the domain. The ECDF used for delaying a packet is chosen depending on the ingress and egress nodes on which the packet enters and leaves the domain, respectively. Alternatively, we can also use a single ECDF irrespective of a packet's ingress and egress nodes. A big advantage of using empirical ECDFs is that we can directly use delay measurements from a real network to configure a domain model.

Inter-Domain Link Models The bottleneck link models between domains are called *inter-domain link models*. This is where packet loss and queuing delay are simulated. The parameters of an inter-domain link model are similar to those of a link in a packet-based simulator, namely link capacity and buffer size. Nonetheless, inter-domain link models are not event-driven. Instead they use the analytical $M/M/1/K$ queue (i.e. a queue with Poisson arrival and service processes and a finite buffer) to model the queuing behavior of the link. Packet drops and queuing delay of the queue depend on the estimated traffic load on the queue. A special case are the links located at the "borders" of the simulated network. They connect the network model to the network simulated using packet-based models.

Application Traffic Models Traffic in this network model is generated by *application traffic models*. They take the form of function that return the generated load at a (monotonously rising) point in time. This may comprise models for VoIP, Video, HTTP, etc. Due to the coarse-grained nature of the network model the focus of these traffic generators lies on aggregates of single sources. For example, an application traffic model may mimic the load generated by a large number of HTTP sessions. Another possibility is to use a trace-driven application traffic model that reads the load at a given time from an external trace file. This is especially useful if we want to feed the simulator with measurements made on a real network. A special kind of application traffic model is used at the connection points between the network simulated by our abstract model and the network simulated using packet-based models. Here, we employ *rate estimators* to convert packet arrival events to the load values used inside our network model. Rate estimation is done by calculating the average bitrate of the packets arriving in a time window. A drawback of application traffic models as used in our abstract network model is that they do not support adaptive traffic sources like TCP because of the lack of feedback mechanism. Nevertheless, it is possible to faithfully simulate TCP if we place the TCP sources and sinks in the packet-based part of the simulation scenario. Our network model will simulate packet loss and delay, causing the TCP endpoints in the packet-based part to adapt their rate. This in turn causes the rate estimator to change its estimate. The accuracy of this approach naturally depends on the length of the rate estimator's time window. If the window is too large, the resulting smoothing effect will distort the TCP simulation.

Multi-Domain Models *Multi-domain models* combine the domain, inter-domain link, and application traffic models of a network scenario and calculate their interactions. Given a time t they compute the load on the single inter-domain links and the resulting end-to-end packet loss ratio and delay. When packet traverses the multi-domain model, the rate estimator responsible for the specified ingress link gets updated first. Then, the multi-domain model inspects the state of the network, updating it if necessary, and calculates both the chance of packet loss and the delay distribution. A pseudo random number generator decides whether or not the packet should be dropped given the calculated drop chance. If the packet is not dropped, another random number distributed according the end-to-end ECDF of the packet's path determines the packets delay. It would be possible to write a simulator based solely on a multi-domain model by inspecting and updating the system state in regular intervals. However, for several reasons it is desirable to combine these models with packet-based simulation. The behavior of an individual flow is easier to describe as a packet-based model, and many protocol and application models already exist for packet-based simulators. Furthermore, a combination of fine grained packet-based simulation and coarse grained analytical models could be very useful in scenarios like the above example of a multi-site virtual private network.

The simulation method used by our multi-domain models employs a combination of elements from fluid simulation and analytical queuing networks, namely the application traffic model and the inter-domain links, respectively. Furthermore, the periodic inspection of the network state is related to time-stepped fluid simulation. While fluid simulation efficiently models the behavior of large traffic aggregates, its assumption that packets arrive in deterministic intervals may be quite unrealistic. Using analytical queue models that assume Poisson arrivals we add non-determinism without requiring changes to the model. On the other hand, Poisson arrivals are generally viewed as too simple to model large traffic aggregates on the Internet. This view stems from a pure modeling point of view. Poisson processes are not suitable to model a traffic aggregate's packet inter-arrivals since Internet traffic has been shown to be far burstier. Other approaches, such as Batch Markovian Arrival Processes (BMAPs) are a much better choice. However, our approach does attempt to model the long term developments of traffic aggregates. Instead, it models the queue state at an arbitrary moment in simulation time based on a given load and link capacity.

In the following sections we will describe the models and algorithm used in our network model in greater detail.

4.2.1 Multi-Domain Models

The purpose of a multi-domain model is to organize and control domain models, inter-domain link models, and application traffic models to form a single network model. Thus, the basis of a multi-domain model is a set of such models and their parameters, e.g. delay characteristics for domain models, link capacities and buffer sizes for inter-domain link models. In order to combine these models to form a multi-domain model we must know how they are interconnected. The topology of a multi-domain model is a directed graph where the domain and application traffic models are the vertices and theinter-domain link models are

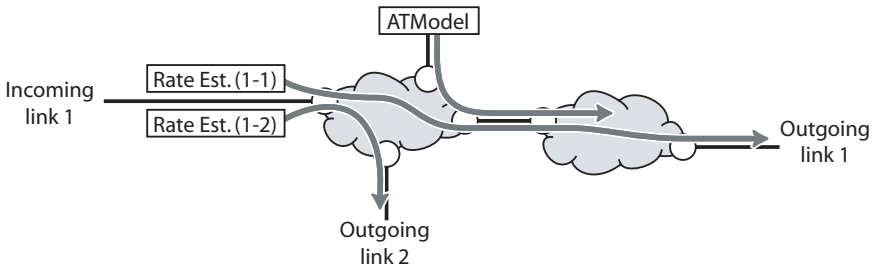


Figure 4.4: Routing paths in the multi-domain model's topology

the edges. Links are always simplex; duplex links can be created by combining two simplex links). This topology graph can be stored in standard ways such as vertex and edge tables.

In order to map the load generated by application traffic models to the right inter-domain links we define a *path* for every application traffic model in the network. A path consists of the traffic model and a sequence of references to inter-domain links and domain models representing the route the traffic takes through the network. Figure 4.4 shows the example of a topology with one incoming link and two outgoing links connecting the multi-domain model to the packet-based model. Rate estimators compute the load on both possible routes, one for each outgoing link. A further application traffic model generates additional load. Note that the path of a normal application traffic model can end anywhere in the simulated network. However, it is not possible to convert the load values of application traffic models to packet arrivals in the packet-based model.

Multi-Domain Load

The load an inter-domain link receives depends on two factors: the load originally generated by the application traffic models, and any potential packet losses in upstream links that reduce the load. It is the task of multi-domain models to simulate the propagation of network load from the application traffic models to the inter-domain link models along their path, considering any packet loss along that path. This is achieved with the following algorithm.

We inspect the network at simulation time t. Let the sequence of inter-domain links $P_s = \{L_1, L_2, \ldots\}$ be the routing path for the traffic originating at the application traffic model s. If link L_i directly follows link L_j on a path we call L_j a *predecessor* of L_i. $s(t)$ denotes the generated load of s at time t. The amount of traffic an inter-domain link L forwards is a function of the offered load λ, written as $L(\lambda)$. We assume that $0 \leq L(\lambda) \leq \lambda$. The exact calculation $L(\lambda)$ will be discussed later. Note that the offered load λ is the sum of loads of all the flows sharing the link.

As long as a traffic flow does not share links, the loads of the links $L_a, L_b, \ldots$ it passes can be calculated by the sequence

$$\lambda_{L_a} = s(t), \quad \lambda_{L_b} = L_1(s(t)), \quad \lambda_{L_c} = L_2(L_1(s(t))), \quad \ldots \tag{4.1}$$

However, as soon as the flow encounters a shared link we need the to know the loads of all other flows on the link to calculate the link's offered load. Conse-

quently, step 1 of the algorithm is to compute the independent sequences of all flows until they meet a shared link. At this point (step 2) we can compute the offered loads of all these shared links. The offered load λ_{L_i} on a shared link L_i is given by the sum of the forwarded loads of all predecessors. If the last calculated element of the path's load sequence was λ we can now calculate the next element with

$$\lambda' = \frac{L_i(\lambda_{L_i})}{\lambda_{L_i}}\lambda\,. \tag{4.2}$$

We include this information in the calculation of each sequence and go back to step 1. This procedure repeats until all paths have been followed to their end. Then, the offered and forwarded loads of all inter-domain links are known.

The above algorithm may be optimized in several ways. First, when updating the system we only have to pursue changes in the offered load as far as they make a difference for the whole system. For example, if a traffic model overloads the first link on its path on one update, any additional load in the next update will influence only this first link. The forwarded load of this link remains the same. Furthermore, changes in the offered load may be marginal, in which case we can ignore this change at the cost of reduced accuracy. We use the following strategy: If the change in offered load of an application traffic model remains below a threshold we ignore the change and do not update the load distribution in the network. However, in order not to accumulate errors we force updates in regular intervals.

Multi-Domain Loss and Delay

Using on the load distribution calculations above, we can find the delay distributions and packet loss ratios of a multi-domain model's paths. The packet loss ratio along the path $P = L_1, \ldots, L_n$ is given by

$$1 - \prod_{i=1}^{n}\left(1 - \frac{\lambda_{L_i} - L_i(\lambda_{L_i})}{\lambda_{L_i}}\right) \tag{4.3}$$

where λ_{L_i} is again the offered load on link L_i.

Delays along a path are modeled in a similar way. The time it takes for a packet to traverse a domain or an inter-domain link can be described as a random variable. Let δ_L be the random variable of the delay caused by inter-domain link L, and let $\delta_{L,K}$ be the random variable of delay in the domain between the inter-domain links L and K ($\delta_{L,K}$ is only defined if L is a predecessor of K). Then, the delay distribution on the routing path P is given by

$$\delta_P = \sum_{i=1}^{n}\delta_{L_i} + \sum_{i=1}^{n-1}\delta_{L_i,L_{i+1}} \tag{4.4}$$

In order to simulate the delays and possible drops of packets traversing the network modeled by the multi-domain model we need to generate random values accordingly. A packet will be dropped if a random value, uniformly distributed on the interval [0,1], is smaller than the packet loss ratio from (4.3). Simulating the delay caused by traversing a path through the simulated network is slightly more complicated. In order to simulate the delay from a single domain or inter-domain link model, we generate a random value that follows the model's

ECDF. The end-to-end delay of a routing path can be simulated by summing up generated random values for each model along the path. The fact that the delay distributions of domain models do not change can be used to make this procedure much more efficient, however. Since these delay distributions are discrete, their distribution functions can easily be convoluted into a single one, which reduces the task of simulating the domain delays to the generation of a single random value. The convolution can be performed efficiently by using the Fast Fourier Transform algorithm [21]. The delay simulation for the domain models on a path can thus be done in a single step. Convolving the link delay distributions is not efficient in normal scenarios as they change rather rapidly according to the load distribution. Modeling a path's delay as a random variable with known distribution has a further advantage. It allows to easily calculate moments like the mean delay or the path's jitter, which would be $\mathrm{Var}(\delta_P)$ if interpreted as delay variation. Even though this does not affect the simulators integration into ns-2 it could be put to use in a stand-alone version of the simulator.

4.2.2 Domain Models

Domain models represent network "clouds" in a simulation scenario where no congestion occurs. The partitioning of a topology into domains and inter-domain links can be freely configured but it must be chosen such as to satisfy this basic assumption of congestion-free domains as closely as possible. "Clouds" of nodes under a common management (e.g. an ISP network) are good candidates, since with policing and shaping performed at the edge routers, congestion within the domain can be avoided. The chosen abstraction allows that domain models only simulate the stationary delay behavior of a network cloud and do not react to changes of network load. Domain models are black boxes; their interior structure is not explicitly modeled. The highest level of detail in a domain model is the distinction of paths through the domain. A domain model with n edge nodes can thus contain $n(n-1)$ delay models, one for each ingress-egress node combination. In simpler cases we can use a common delay model for all paths through the domain. This becomes useful if we lack the information to create a more complex model. Using simple models can also significantly reduce the memory consumption of a simulation.

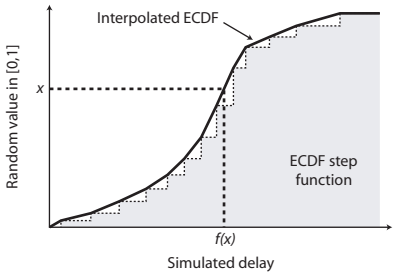


Figure 4.5: Generating random values using an interpolated ECDF

During preliminary evaluation we found that in many cases empirical cumulative distribution functions (ECDFs) are a good method to model the delay behavior of network domains. They can be easily built from a series of delay

measurements taken from a real network. In the optimal case, one-way delays should be used, but since this requires clock synchronization of the measurement endpoints we might have to approximate them by using round-trip time measurements divided by two (which may be a bad approximation if the paths are strongly asymmetric). For the integration with packet-based simulation we need to be able to generate random values based on the ECDF. A basic approach is to store the observations in a table and then randomly select table entries using a uniform distribution. Given a sufficient sample size, this approach yields very good results. However, large tables may have a severe impact on the memory consumption of the simulator. Fortunately, we can usually use linear interpolation to reduce the size of the tables with only minimal additional error. The procedure can be seen in Figure 4.5. We start by generating a random value x, uniformly distributed on the interval $[0, 1]$, which designates a position in the sorted observation table (seen as a step function in the Figure). The two nearest observations are then interpolated to get a simulated delay value $f(x)$. It is important to note that ECDF models, while giving good reproductions of observed first and second-order moments in measurements, ignore any non-stationarity of the sample.

4.2.3 Inter-Domain Link Models

Inter-domain link models cover the dynamic parts of network behavior, like the effect of queuing and congestion on delay and packet loss. Since they represent a single physical link between the interfaces of two nodes it is an obvious approach to model them as an analytical queues. We chose the simple $M/M/1/K$ queue as an approximation, that is, a queue with Poisson arrival and service processes, a single server (the physical link) and system capacity K. The arrival and services rates λ and μ are given by the link's offered load and capacity, respectively. The system capacity K can be set to a typical value (e.g. 128-packet buffers are rather common in routers).

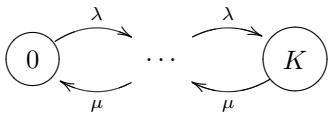


Figure 4.6: Birth and Death Process

In order to model the behavior of the inter-domain link we have to find the probability p_i of the system to be in state i, where state K means the queue is full, and state 0 means the system is empty and does not send. The $M/M/1/K$ queue is a birth and death process as shown in Figure 4.6. For a birth and death process of this kind the probabilities p_i are given by

$$p_i = \begin{cases} \frac{1-\lambda/\mu}{1-(\lambda/\mu)^{K+1}}\,, & i = 0 \\ (\lambda/\mu)^i p_0\,, & i > 0 \end{cases} \tag{4.5}$$

if $\lambda \neq \mu$, and

$$p_0 = p_1 = \ldots = p_K = \frac{1}{K+1} \tag{4.6}$$

if $\lambda = \mu$. Since p_K is the probability of the system being full it is also the loss ratio of the link. The functional representation $L(\lambda)$ of the inter-domain link used in the computation of load in the network can thus be written as $L(\lambda) = (1 - p_K)\lambda$, with p_K calculated according to formulas 4.5 and 4.6. From the probabilities p_i we can further construct a discrete density function of the link's delay distribution. The number of bytes that are in the system when another byte arrives is proportional to the time this byte has to wait before it is sent to the link. δ_{pr} is the propagation delay on the link, which depends on physical properties of the link, e.g. its length. The discrete delay distribution thus looks like this

$$\begin{pmatrix} p_0 & \cdots & p_{K-1} & p_K \\ \delta_{pr} + \frac{1}{\mu} & \cdots & \delta_{pr} + \frac{K}{\mu} & \infty \end{pmatrix} \tag{4.7}$$

The infinite delay in the case of a full queue indicates that this packet is effectively lost.

4.2.4 Hybrid Simulation

We enhance the traditional, packet-based simulator ns-2 by enabling its nodes to contain analytical multi-domain models. A single simulator node can stand for, and behave like, a whole subnetwork. When a simulated packet reaches an enhanced node, it triggers an inspection of the underlying multi-domain model to determine how much the packet should be delayed and whether it should be forwarded at all. Both decisions are based on the cumulative forwarding probability and delay distribution calculations described in the above sections.

Packets flowing through the simulated network add to the load on the respective path. In our network model this effect is modeled by rate estimators that converts packet arrival events into a bitrate estimate for each routing path between an ingress and an egress link of the multi-domain model. A good way to estimate bitrates from packet arrival events is to use a sliding time window algorithm. The number of bytes received in the time window Δt is added up and divided by Δt. We have implemented this by keeping a fixed-size list of recently arrived packets (we use a 128 packet buffer as a default). Therefore, the length of the time window changes depending on the packet rate. High packet rates result in a short time window while low packet rates let the window grow, up to a configurable bound. There are two reasons for not choosing a fixed-size time window: First, the number of packets in a a fixed-size window does not have a natural upper bound. Thus, memory consumption might become a problem for high bandwidth scenarios. Second, fixed-size windows cannot accurately estimate low packet rates, because if the inter-arrival time of packets is smaller than the window size, the estimated bitrate will alternate between two values depending on whether or not there is currently a packet inside the time window. If the window grows bigger with lower packet rates, we can reduce this undesirable behavior.

While packets generated in the event-driven simulator may influence the network models inside enhanced nodes, the load generated by application traffic models do not create additional packets outside of the enhanced node. Our approach only allows packets to go through enhanced nodes, not to be created by them.

4.3 An Extension Mechanism for the ns-2 Simulator

4.3.1 The Extension Module Interface

In order to combine the packet-based approach of ns-2 with other simulation methods we extend the ns-2 nodes with arbitrary simulation modules. *Module interface* units in modified nodes load simulation modules during run-time using the dynamic loadable library (DLL) mechanism available on most modern operating systems. After loading, the interface unit tells the module to initialize itself and asks it to create a *simulation module object* (internally called `ISP_module`, because of the original focus of the simulator on inter-domain networks). Subsequently, the extended node will notify the simulation module object about every arriving packet and in turn receive a floating point value determining whether to drop or to delay the packet. Positive values indicate that the packet should be delayed by this many seconds; negative values cause the extended node to drop the packet. If the simulation module object returns zero the packet will be forwarded without any modification.

```
ISP_module *isp_module_init(int node_id);

class ISP_module
{
  public:
    virtual double process_packet(int prev_node_id, int next_node_id,
                                  double time, struct ISP_pinfo *p);
    virtual int command(const int argc, const char*const* argv);
};
```

Figure 4.7: The interface used by simulation modules

Modules for our extension mechanism have to implement the simple C++ interface shown in Figure 4.7. The `init()` function is always called when a module is loaded. It performs any necessary initializations and creates the simulation module object through which the extended node communicates with the encapsulated simulation model.

During a simulation run, the node notifies the simulation module object about every arriving packet, its time of arrival, and its incoming and outgoing links using the `process_packet()` method and in turn receives a floating point value indicating whether to delay or to drop the current packet. It is left to the module to convert the packet arrival events into an internal representation suitable for the encapsulated abstraction. The ISP_pinfo structure passed to the `process_packet()` method comprises the source and destination addresses of the packet, its size in bytes, its time-to-live (TTL) field, and its ns-2 packet type.

Loading and configuration of simulation modules can be done from within ns-2 Tcl scripts. Extended nodes provide the method `attach-module`, which takes the path to the module file as argument and and tells the integrated extension module interface to load the module. Each module may define its own set of

command available through the extended node's `command` method. This facility is mostly used to configure the module. For example, the call `modcmd config configuration.xml` might load the file `configuration.xml` and configure the module according to its contents. Since these Tcl methods may be called at any time during a simulation run, modules must be ready to be dynamically reconfigured or even replaced by other modules.

4.3.2 Modifications to ns-2

Adding the extension mechanism discussed above requires several modifications to the ns-2 code, especially to the ns-2 node. The original structure of an ns-2 node was discussed in Section 2.2.1.

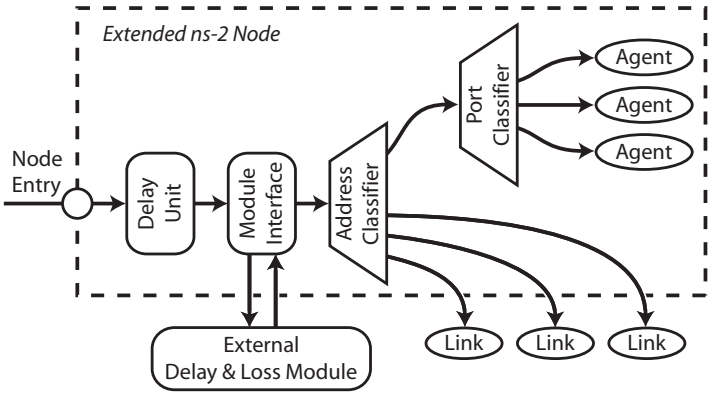


Figure 4.8: Structure of an extended ns-2 node

Figure 4.8 shows the modifications necessary to extend an ns-2 node. Arriving packets now go through two additional steps before they reach the address classifier: the *module interface*, which provides the mechanism to load external simulation modules into the simulator, and the *delay unit*, which is an additional, simple means to model processing delay in the node.

The focus of this chapter is on the interface unit, which delegates the simulation to an external delay & loss module providing the interface presented in Section 4.3.1. In addition to providing the simulation module object with information about the packet and the current simulation time it must also tell it on which link the packet arrived and on which link it will leave the node. Knowledge about the incoming and outgoing links of a packet is indispensable if the external module is to simulate a sub-network with multiple border links. Without it the module could not determine the route of a packet through the sub-network.

It is simple for the extension interface to determine the outgoing link of a packet. It can just consult the routing table of the extended node's address classifier to determine to which link the packet will be forwarded once it has been processed by the extension module. Unfortunately, there is no easy way for the extension interface to identify the link through which a packet has been received. The reason for this is that ns-2 objects use the `recv()` method of other ns-2 objects to hand packets over to them, and `recv()` provides no way to specify the caller of the method. Therefore, a packet that reaches the interface

```
# Load modified constructors
source ispns.tcl

# Create the simulator, node, and link objects
set ns [new Simulator]
set n0 [$ns IM-node "Node-0"]
set n1 [$ns IM-node "Node-1"]
set n2 [$ns IM-node "Node-2"]
$ns IM-simplex-link $n0 $n1 1Mb 19ms DropTail
$ns IM-simplex-link $n1 $n2 1Mb 19ms DropTail

# Attach a module to the middle node, configure it,
# and turn on tracing
[$n1 entry] attach-module example.mod
[$n1 entry] modcmd delay 0.1
[$n1 entry] trace
```

Figure 4.9: Example Tcl script loading an example module into an ns-2 node

module could have been received on any link connected to the node. For a normal ns-2 node this is usually not a problem since it does not need to know the incoming link in order to route a packet. An extended node however needs this information. We solve this problem by adding a new field to the packet class of ns-2 and letting the nodes write an identifier into this field whenever they forward a packet. This field can later be inspected by the extension interface of a node. Additionally, we create an object reference table for all nodes and links in the simulator, which is used together with the identifier field to identify the incoming link of a packet. The object reference table must be updated whenever we create a new node or link. Special constructors in the Tcl front-end of ns-2 take care of this. `IM-simplex-link` and `IM-duplex-link` accept the same parameters as the original link constructors. They build a normal ns-2 link like the original constructor would, but they also create an entry in the object reference table. The `IM-node` constructor additionally extends the Tcl node object it builds with the methods necessary to load and configure extension modules, `attach-module` and `modcmd`.

In addition to the module interface unit extended nodes also contain a *delay unit*, which provides a simple way to add a measure of randomness to an ns-2 scenario without having to implement an extension module. When it receives a packet it waits for a random amount of time before it forwards the packet to the module interface. The distribution of the additional delay can be either Gamma or Gauss, or the delay can follow an empiric distribution. The latter is used to mimic the distribution of measurements from a real device and can be configured using histogram files. The delay unit may be used to simulate stationary processing delays. However, it cannot simulate non-stationary effects (such as load-dependent latency). Also, the delay distribution applied to packets is always the same, irrespective of the packets' incoming and outgoing links.

An example script using the module interface facility can be found in Figure 4.9. It creates a simple scenario of three nodes connected by two simplex links. The first instruction loads the modified constructors for node and link objects into the Tcl interpreter. Then, these constructors are used to set up a

simple three node topology with two simplex links. Finally, we load and configure a simple example module into the middle node using the new `attach-module` and `modcmd` commands. The instruction "`delay 0.1`" causes the example module to delay all passing packets by 0.1 seconds. The `trace` command tells the extension interface to write events caused by the module to the simulator's trace file.

4.4 Integration into a Network Monitoring and Prediction Toolkit

We have developed the hybrid simulator in the course of the of the European Union research project Intermon. This project aimed at creating an integrated architecture for the study of inter-domain networks. The architecture includes tools for various aspects. Traffic meters and topology detectors gather information about the network and make it available to a set of modeling tools. The models thus created from network measurements can be further used to build simulation scenarios for on of the four integrated simulators. The data resulting from measurement, modeling, and simulation can then be visualized using the integrated visual data mining system.

In this section we describe the our integration of the hybrid simulator and the analytical extension module described in Sections 4.3 and 4.2 into the Intermon architecture.

4.4.1 Architecture overview

The Intermon architecture has been designed as a highly distributed system. Measurements, data storage, simulation and visualization can all be hosted on individual systems. The communication elements are implemented in Java using the Java Messaging Standard API (JMS) as a communications middleware. User requests are handled centrally in the *global controller* (GC). Depending on the type of request, the GC sends generic requests to specific *tool managers* (TMs). Those are responsible for converting the generic requests into format the tool understands. Tool managers convert the tool's output back into a format that the GC understands and return them. Due to their complexity, simulation tool have a specialized variant of tool manager called *simulation manager*.

Many requests a user may send are of a long-term nature, for example requests initiating inter-domain monitoring on a set of network links. This calls for an asynchronous communication scheme. When a tool manager receives a request it may either return the results immediately, or it may return a "work in progress" message and send the results as soon as they are available.

The user can build simulation scenarios using a graphical interface and then click a button to submit them. Once a simulation request has been submitted it is sent to the GC, which in turn forwards it to the appropriate simulation manager. The GC may also check or modify the request. For example, it may query the global database to retrieve additional information for the specific simulator and add this information to the request. Once the simulator has finished processing, the GC sends a notification to the user who initiated the request, containing information about the location of the results (depending on the simulator they may have been sent to global database or stored in a remote

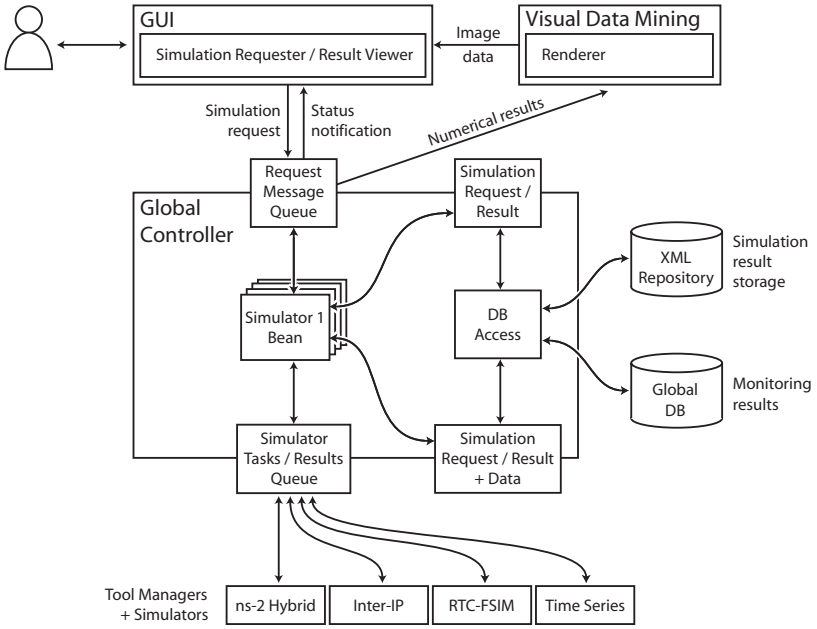


Figure 4.10: The part of the Intermon architecture concerned with simulation

repository). The user can then request further analysis of the simulation results, which usually involves the preparation of images by Intermon's Visual Data Mining (VDM) toolkit.

4.4.2 Generating and Processing Simulation Jobs

A generic XML-based simulation job description format has been developed to control the different integrated simulators in a common way. This format separates the data into sections containing simulator-specific data and sections with simulator-independent data. Simulation job descriptions are structured as follows:

Topology: The simulator-independent topology contained in simulation job is usually a BGP topology based on data previously collected by the Inter-Route tool [43], which queries BGP data and reconstructs the topology based on route advertisements and route withdrawals.

User applied changes: After selecting a BGP topology the user can make changes to it. These changes will be encoded in XML and listed here. Since each of the available simulators supports different kinds of actions, the exact format of this part depends on the selected simulator. Possible changes include removing a link, changing its capacity, or adding flows to the scenario, amongst others.

Simulator specific parameters: Each simulator has some simulator specific parameters that are not related to the topology. Those are encoded within this section, enabling the user to supply general instructions and param-

eters to the simulator. Examples of such parameters are the duration or the granularity of the simulation.

```
<simrequest>
      <BGPTopologyTree>
            BPG topology from the topology detection tool
            <AS>
                  AS description, possibly including recent BGP
                  path updates
            </AS>
            ...
            <Link>
                  Inter-AS link description
            </Link>
            ...
      </BGPTopologyTree>
      <changes>
            Simulator dependent changes/additions to the scenario
            <action type=''set_link_capacity''
                    src=''1111'' dst=''2222''>
                  1M
                  Example action changing the link capacity between
                  ASs 1112 and 2222
            </action>
      </changes>
      <params>
            General parameters for the chosen simulator
      </params>
</simrequest>
```

Figure 4.11: Structure of a simulation request

The XML job description is generated by the graphical user interface and sent via the global controller to the tool manager of the appropriate simulator. Figure 4.11 shows a simplified XML simulation request as it would be sent to the simulator.

4.4.3 The Tool Chain

Since network simulations can require a lot of computing power and also can produce a huge amount of data, scalability was a central aspect during the design and the implementation of the hybrid simulator's tool manager and its underlying tool chain (see Fig. 4.12). The system supports the processing of simulation requests in parallel and thus supports multiprocessor computers as well as computer clusters. Naturally, the simulator itself is not made parallel by this mechanism. Parallelism can thus only be exploited if the simulation server processes several simulation requests concurrently.

On top of the tool chain that processes the simulation requests resides the tool manager, which handles communication with the global controller. When

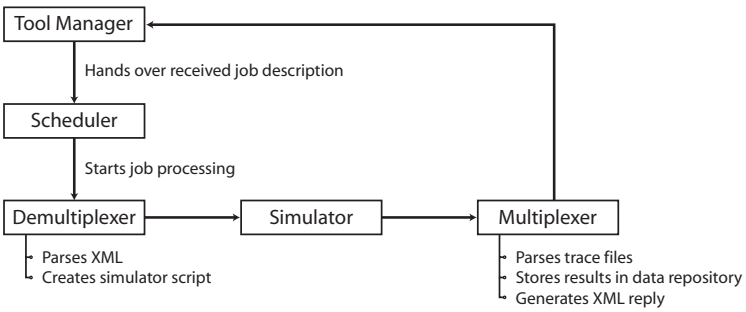


Figure 4.12: The hybrid simulator's tool chain for the automatic processing of simulation requests

it receives a simulation job, the tool manager automatically returns a short message to the global controller signaling the graphical user interface that the job was received and will be processed. After that the tool manager hands the job to the scheduler, which stores the job description into a temporary directory on the disk and then initiates a tool chain process for that directory. After doing this the scheduler does not have to wait until the tool chain finishes but is immediately ready to receive further job descriptions. The scheduler may also decide on which computer/processor a particular job should be processed and therefore can provide flexible load balancing.

The first steps in job processing consist of converting the XML simulation job description into configuration files for the hybrid simulator, and then starting the simulator. This is done by the demultiplexer component. After the simulation has finished, the multiplexer components takes care of processing the simulator tracefiles and stores the results in a data repository. Finally, it returns a message telling the system by which URL the results can be obtained.

The messages returned by the hybrid simulator's tool manager include some descriptive comments on the content of the reply and indicate the processing time and the status (e.g. running, ok, canceled, broken) of the simulation job. As the simulators may produce several different types of results, multiple result sections may be included in the reply, one for each desired measure (e.g. time series of throughput, or of end-to-end delay on a certain path). The sub-results have their own descriptive comment and a type identifier to help visualization.

Using a data repository is necessary since simulation results may grow too large to be processed directly by the messaging system. Due to performance reasons, the data repository is usually implemented as an FTP or HTTP server located on the simulation computer/cluster. Alternatively, for small amounts of data, the results may also be returned directly within the simulation reply.

4.5 Tests and Experiments

4.5.1 Domain and Inter-domain Link Model Tests

The hybrid simulator divides a network scenario into partitions that are assumed to be free of congestion (called domains), connected by possibly congested inter-

domain links. Consequently, the accuracy of the approach depends to a large extend on these two kinds of models.

Since domain models represent network areas without congestion they may restrict themselves to simulating stationary delay usually based on delay measurements from the real network. In order to evaluate the performance of a domain model we can thus compare the measured delays we used for creating the model to the simulated delays it generates. We have made delay measurements between a computer at the University of Bern and a computer at the ETH Zürich using `ping`. Both computers were 9 network hops away from each other. The network between the University of Bern and the ETH Zürich is the Swiss scientific network SWITCH, a high bandwidth network that hardly ever suffers from congestion, which makes it a suitable candidate for modeling with a domain model.

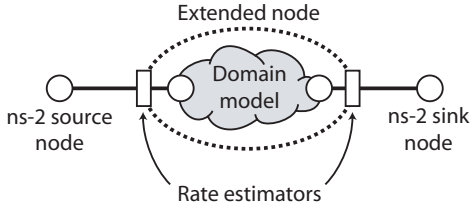


Figure 4.13: ns-2 setup to simulate the delay of a single ISP

The simulation was run using the simple topology shown in Figure 4.13. A constant flow of packets was sent from the source to the sink node within ns-2. For both, measurements and simulation, we have used a packet rate of one packet per second. On their way to the sink the packets encountered an extended node containing a domain model of the network between Bern and Zürich. The border links connecting ns-2 with this domain model were configured with very high capacities in order not to distort the results.

Figure 4.14 shows the the histograms of both the measured and the simulated delays. Both graphs show almost exactly the same delay behavior. This shows that if we want to simulate a congestion-free part of a network, domain models can be a very accurate way of modeling delay.

The packet simulated drop ratio of a flow of ns-2 packets sent through a network modeled by the extension module depends on the accuracy of the inter-domain link models as well as the behavior of the rate estimators. We have evaluated the accuracy of both these elements by comparing the throughput observed in a bottleneck scenario purely based on ns-2 scenario to the packet loss ratio observed in a scenario that models the bottleneck link using an inter-domain link model. The scenario consisted of three consecutive links, of which the middle one was a 2 Mbps bottleneck. We have studied the behavior of this link under five kinds of traffic load: 1 Mbps, 2 Mbps and 4 Mbps CBR traffic, FTP traffic originating at 5 agents on the source node, and a mix of FTP traffic from 3 sources and 1 Mbps CBR traffic. The topologies used in this experiment are shown in Figure 4.15.

Figure 4.16 shows a comparison of the transfer rates achieved with both topologies. In all three CBR scenarios the throughput observed with the pure ns-2 topology was very similar to the throughput achieved with the inter-domain

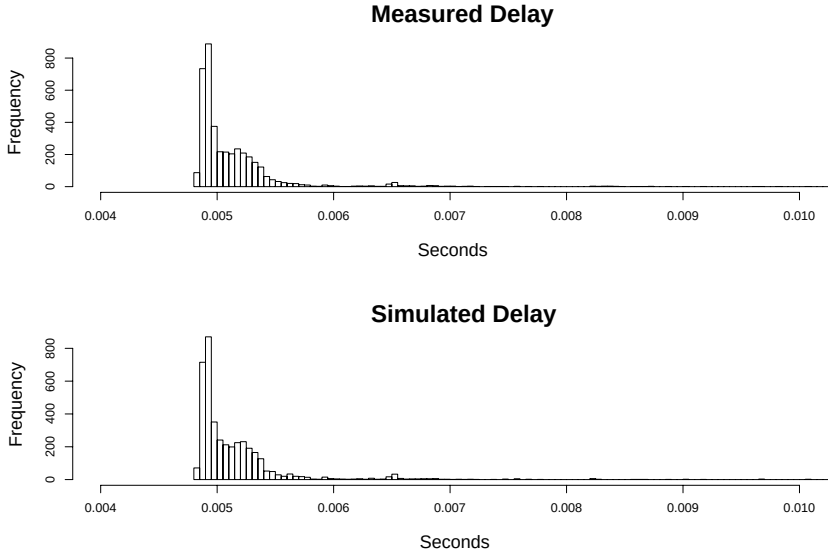
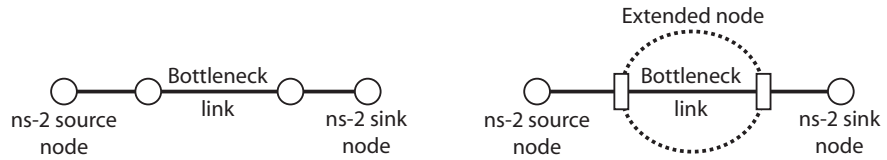


Figure 4.14: Delay histograms from measurements (upper graph) and simulation (lower graph)



(a) Pure ns-2 scenario (b) Hybrid scenario

Figure 4.15: Simulation topologies for evaluating the throughput of inter-domain link models

link model. Also in the FTP scenario we observe only a minimal difference. The mixed scenario with CBR and FTP traffic appears to be more problematic. With both, the ns-2 link and the inter-domain link model, the transfer rate fails to reach the bottleneck capacity. The TCP-based FTP flows seem to be unable to exploit the full 1 Mbps capacity available to them. However, in the hybrid topology they appear to fare better than in the pure ns-2 topology. We believe this is due to the stochastic nature of dropping in the analytical inter-domain link model. TCP reacts strongly to correlated drops. If it detects two consecutive packet drops it halves its sending rate. The fact that the inter-domain link model drops packets randomly makes this situation less likely in the hybrid topology. This effect is similar to the behavior of random early detection (RED) queues [30], which are known to enable higher transfer rates with TCP than traditional drop-tail queues do. RED queues also drop packet randomly when the buffer starts to fill up.

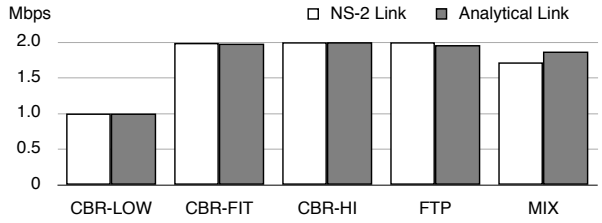


Figure 4.16: Comparison of an ns-2 to an analytical link: transfer rates

We conclude that inter-domain link models accurately can model throughput and packet loss in many scenarios. However, the throughput achieved by TCP flows may get overestimated in cases where correlated packet drops are frequent.

4.5.2 Scalability Test

In order to demonstrate the scalability of our inter-domain simulation approach we have created two ns-2 scenarios of different complexity. They are shown in Figures 4.17 and 4.18. In the small scenario a G.711 voice stream between the nodes $n1$ and $n4$ was used as reference stream. FTPDATA flows were configured between the nodes $n2.1$ and $n3.2$ as well as the nodes $n2.2$ and $n3.1$. The middle link is the bottleneck in this scenario.

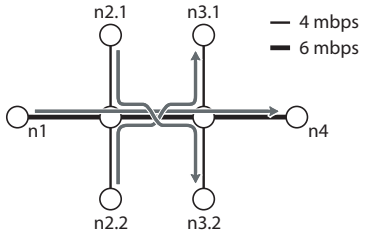


Figure 4.17: Scalability test – small scenario

In the big scenario shown in Figure 4.18 the topology and the traffic flows are more complicated. Again, G.711 reference traffic was transmitted between $n1$ and $n4$. Additionally, each of the leaf nodes on the left sent an FTPDATA flow to a leaf node on the right, determined by the following pattern: if the source node's ID is $n2.1.x.y$ it sends traffic to $n3.2.y.x$. Conversely, if its ID is $n2.2.x.y$ it sends traffic to $n3.1.y.x$. This scheme results in a fairly complicated cross-traffic pattern. Again, the bottleneck in this scenario is the middle link.

We have modeled both scenarios using a multi-domain model (Figure 4.19) consisting of two domain models A and B and an inter-domain link model as a bottleneck between them. For the small scenario the domain models were configured to have zero delay since the ns-2 nodes they represent do not model any delay either. For the big scenario the domain models were configured using delay distributions extracted from the ns-2 simulation traces. A represents the source nodes and B represents the sink nodes. This mapping is possible since

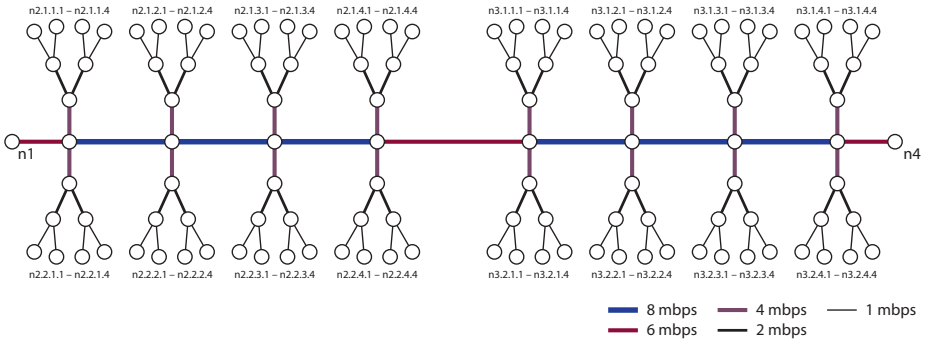


Figure 4.18: Scalability test – big scenario

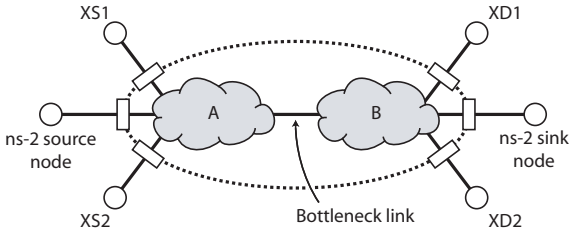


Figure 4.19: Scalability test – inter-domain model scenario

there is no bottleneck link in these groups of nodes. In order to add some dynamics to the scenario, we configured ns-2 generated FTPDATA cross traffic from $XS1$ to $XD2$ and from $XS2$ to $XD1$.

We ran both scenarios in ns-2 and compared simulation run time to the time required to simulate the inter-domain model scenarios. The results are shown in Figure 4.20. We observe that the small scenario and took virtually the same amount of time in the original ns-2 simulator as in the hybrid simulator. Both simulators used around 20 seconds. The big scenario however takes much longer to simulate in ns-2 while the hybrid simulator requires only slightly more time to simulate it. This is no surprise considering the very similar multi-domain models for both scenarios. Nevertheless, these results demonstrate the scalability gain that can be achieved with suitable abstract scenarios.

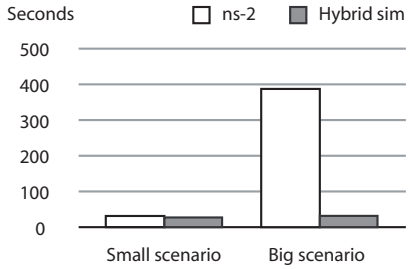


Figure 4.20: Run times observed in the scalability test

4.5.3 Comparison to Testbed Measurements

In order to evaluate the performance of the simulator compared to a real network we set up a testbed experiment using the extended cross traffic topology shown in Fig. 4.21. All nodes were Intel-based Linux systems, interconnected by 100 mbit ethernet links.

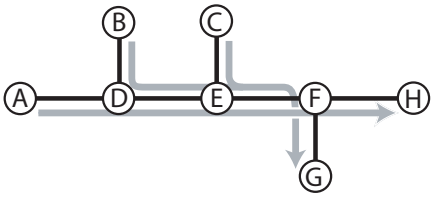


Figure 4.21: Testbed topology

Three traffic flows were sent through the network: one from node A to H, one from B to G, and one from C to G. We used the flow from A to H as reference for comparison with the simulator results. Each of these flows consisted of several 1 mbit/s CBR sub-flows, which entered the network with Pareto inter-arrival times and remained for an exponentially distributed amount of time. The traffic patterns were precomputed and stored in trace files, one for every traffic generator. During the experiment the traffic generators would then read these files and generate traffic accordingly. Figure 4.22 plots the traffic loads produced by each of these sources. For top graph displays the sum of the three individual flows. When it rises beyond the theoretical maximum link capacity indicated by the dashed line we can expect queues to build up. This will lead to increased delay and packet loss.

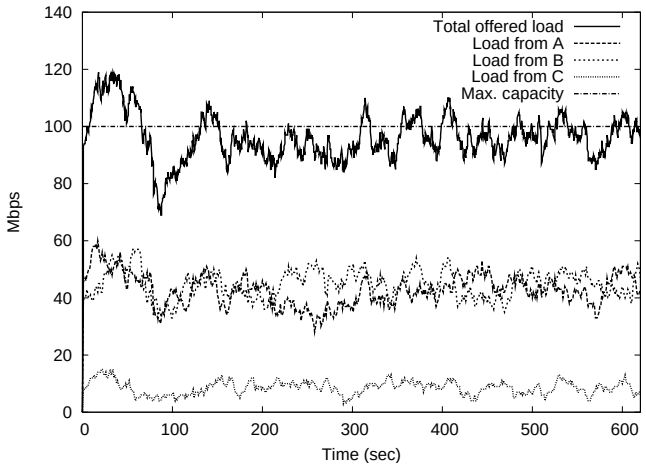


Figure 4.22: Generated traffic in the testbed

The corresponding simulation scenario was configured as follows: The ethernet links were modeled with inter-domain link models of the same capacity. As the delay introduced by the Linux nodes was expected to be minimal we chose to model them by domain models with zero delay. Interfaces to ns-2

were attached at A and H, and a reference stream of packets generated by ns-2 was sent along the path A-D-E-F-H to determine delay and loss ratio. Inside the extension module, trace-driven application traffic models generated traffic according to the same trace files as used in the testbed.

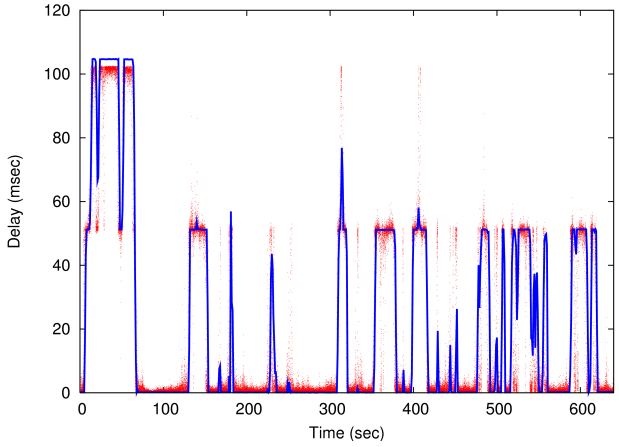


Figure 4.23: Comparison of delays from testbed and simulation scenario

Figures 4.23 and 4.24 show a comparison of measurements and simulation results. The delays in the testbed showed very little variance even with full queues, which is due to the CBR characteristic of the generated traffic. Consequently, the inter-domain link models, expecting Poisson arrivals, overestimated the traffic's burstiness. Nevertheless, the mean of delay was similar in both, testbed results and simulator traces. In order to illustrate this, Figure 4.23 shows the simulated delays as a dot cloud and the measured delays as a line. Both graphs match rather well. Only when nodes E and F are under full load, packet forwarding in the routers begins to slow down slightly, which leads to the small gap between simulated and measured delays.

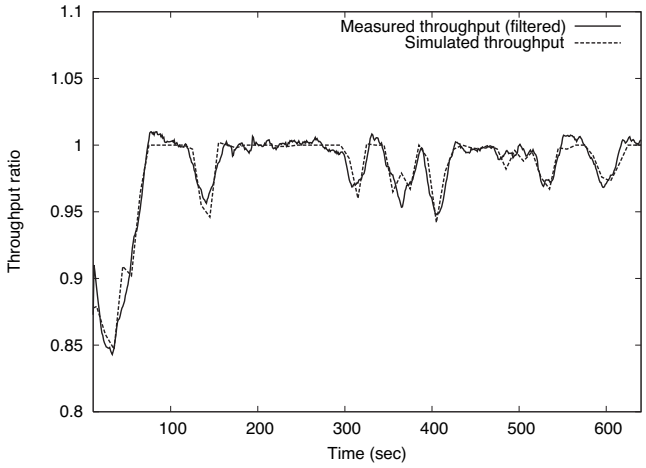


Figure 4.24: Comparison of throughput from testbed and simulation scenario

In contrast to delay the throughput in the testbed proved to be rather bursty. This didn't have much effect on the delay as the effect was hardly noticeable in comparison with queuing delay. However, the resulting small transient queues caused the throughput graph to be rather noisy. The graph in Fig. 4.24 was therefore smoothed using a box filter. Nonetheless, it can be seen that the simulated values closely match the testbed measurements.

Chapter 5

A Peer-to-Peer Distance Prediction Service

5.1 Introduction

A fundamental strategy of the Internet is to move complexity away from the network core towards the network endpoints. Protocols and architectures focus on end-to-end mechanisms for adaptivity and do not require feedback from network-internal devices to detect and adapt to varying network conditions. Peer-to-peer and overlay networks are sophisticated cases of this design approach.

The structure of peer-to-peer networks continuously changes due to nodes joining or leaving the network. They may also dynamically change their structure in order to adapt to changing characteristics of the underlying physical network. This property makes it hard to predict their performance using a simulation approach like the one we presented in Chapter 4. The cost of creating a simulation scenario and running the simulator is too high to do this each time a peer-to-peer network changes its structure.

Nevertheless, peer-to-peer networks still require predictions of the underlying network's characteristics in order to adapt. A simple and wide-spread method is to perform round-trip time measurements between the peer-to-peer nodes and choose the topology based on the results. However, this method may be inefficient because of the potentially large number of measurements. There are numerous architectures and frameworks aimed at optimizing this problem. Distance estimation services provide estimates of end-to-end characteristics based on a small number of measurements. In Section 3.4 we have given an overview of the existing work and its drawbacks, and we have discussed the different aspects we believe are crucial for the success of a given approach. In this chapter we now present a cluster-based end-to-end distance prediction service that was designed to perform well with regard to these aspects. After giving an overview of the service in Section 5.2, we describe the algorithms for the construction of the peer-to-peer structure and for the measurement-based identification of remote clusters in Sections 5.3 and 5.4, respectively. In Section 5.5 we have a closer look at the service's architecture. We present and discuss simulation

results in Section 5.6. Finally, we present the design of a prototype implementation and the corresponding tests in Section 5.7. The work in this chapter was also presented in [118] and[117].

5.2 Overview of the Proposed Service

Basic Approach

We propose a distance measurement and prediction service that provides a common distance measurement API to network applications and reduces the measurements made by coordinating the measurement activities. If an application requests a distance measurement to a known target, the service may return a prediction rather than making an actual measurement. The service architecture is made up of groups of nodes from the same network location. Inside these groups, the nodes form a small peer-to-peer network to exchange and store distance measurements and predictions. The basic structure of this architecture was originally inspired by mOverlay. In [145] the authors propose to use mOverlay groups as the basis for a distance estimation system. However, the group concept is the only part of our architecture that remains similar to mOverlay.

Measurements are mostly obtained by observing the regular application traffic of the group members. Applications may notify the service about network conditions they observe during normal operation. Additionally, network monitors may be used to extract measurements from application flows. If needed the service may also actively measure distances. Nevertheless, we expect that passive measurements should be sufficient in most cases since the measurement requests from clients tend to pertain to the same hosts that the clients communicate with. In other words, the most popular measurement targets we will automatically have the most measurement data available. As such, the basic approach of the service resembles that of a web proxy server.

Although the construction algorithms of the approach focus on round-trip time, it is not limited to that measure and can easily handle other types of measurements. Moreover, because of the amount of available distance measurements, we can employ statistical methods for identifying trends and variances and even calculating predictions of future distances.

Clustering

The number of distinct measurement targets the service has to handle may be quite large. However, we can often find clusters of measurement targets that can be treated as a single unit, normally because they are near to each other in the network topology. This kind of clustering may greatly reduce the complexity of storing and managing measurement data.

We use the following definition of a *cluster*. Let N be the set of endpoints in the network. $d_t : N \times N \mapsto \mathbb{R}_0^+$ is a time dependent *distance function* if and only if, for all $n \in N$, $d_t(n, n) = 0$. Note that, in contrast to Euclidean geometry, neither symmetry $\big(d(n, m) = d(m, n)\big)$ nor the triangle inequation $\big(d(n, m) + d(m, o) \geq d(n, o)\big)$ are required. Given a distance function d we call

Table 5.1: Terms used in our distance prediction service

Cluster	A set of nodes that are near enough to each other in the network topology to be treated as an equivalence class with regard to distance measurements. Nodes belonging to a cluster are not necessarily aware of that fact. We refer to a node's cluster as its *local* cluster. Conversely, we refer to all other clusters as *remote* clusters.
Group	Groups are the basic organizational units of the distance prediction service. They consist of nodes from the same cluster that communicate with each other to coordinate measurements and exchange the resulting data. We refer to a node's group as its *local* group. Conversely, we refer to all other groups as *remote* groups.

$C \subseteq N$ a *cluster with respect to* d if and only if, for all points t in time

$$|d_t(o,n) - d_t(o,m)| < \varepsilon, \quad \forall\, n,m \in C,\ \forall\, o \in N \setminus C \tag{5.1}$$

where ε is a threshold that depends on the statistical error. The endpoint o is called the *observing node*. If two endpoints belong to the same cluster they are called *neighbors*. The sets $\{n\}$ and N are trivial cases of clusters. If $C \subseteq D$, we call cluster C a *subcluster* of cluster D and D a *supercluster* of C.

It is important to note the difference between the terms *cluster* and *group*. We use the term *cluster* for sets of nodes that can be treated as equivalence classes with respect to distance measurements. Nodes belonging to a cluster may not be aware of the fact. Conversely, a *group* is an organizational unit of several nodes that coordinate their distance measurements and exchange the resulting data. This exchange of distance measurements is only possible if all group members belong to the same cluster (see Equation (5.1)). Accordingly, there is a direct relationship between the concepts of clusters and groups. Table 5.1 gives short definitions of both terms for later reference. A practical use of the relationship between groups and clusters is that, since they are clusters, groups can be treated as single units by other groups. Figure 5.1 illustrates the concept. When predicting distances, the local group does not distinguish between remote hosts and remote groups.

We use two distinct algorithms for identifying groups and clusters, respectively. The algorithm used to form groups is described in Section 5.3. In Section 5.4 we present a method to identify neighbors from a single observing node using time series of measurements, and we show how to use this information to build clusters.

Group Organization

Groups are roughly organized as follows. A *group leader* keeps track of the group members and helps joining nodes find an appropriate group. In order not to create a single point of failure several members of the group can be assigned as backup leaders. Normal group members handle the measurement and prediction requests coming from client applications.

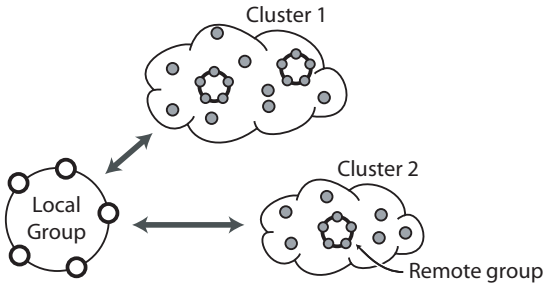


Figure 5.1: Clustered view of the network from the vantage point of a single group

Measurements and prediction models are kept in a distributed repository based on a distributed hash table (DHT) approach. Each group member is responsible for storing the data of a given set of clusters. The DHT serves as a lookup mechanism for finding the responsible group member, given a cluster ID or the IP address of a measurement target. When an group member receives a distance prediction request it first searches the repository for existing information about the target in question. If successful, it returns a predicted distance value and the corresponding estimated error. Based on the estimated error the application may then decide to use the predicted distance or to initiate an active distance measurement.

Advantages of the Proposed Service

The proposed service was designed with regard to the different aspects of distance estimation services discussed in Section 3.4. A main advantage of the approach is that it can provide accurate predictions to hosts anywhere on the Internet even if it is deployed only at a single site. It does not require a large deployment in order to be useful. Other distance estimation approaches only yield meaningful results for distances inside the network area in which they are deployed. Beyond this area, the error grows dramatically. Furthermore, the service's peer-to-peer structure does not require any additional network infrastructure and makes it scalable and robust.

Unlike other approaches, the proposed service not only provides estimates of average round-trip times but is able to compute more sophisticated predictions for any kind of distance measurement. The resulting increased storage and CPU requirements are distributed over the peer-to-peer network.

5.3 Locating Groups

The problem of assigning nodes to groups based on their proximity in the underlying network topology has been addressed by other approaches. Binning [104] assigns nodes to a partition of an n-dimensional space based on their position in the network. IDMaps uses groups of nodes with identical IP address prefixes assuming that similar addresses imply closeness in the underlying network. However, the groups identified by these approaches are rather coarse grained and thus not suitable as basis for our architecture. In a first step we

have considered using mOverlay's group locating algorithm (described in Section 3.3) for our architecture. Unfortunately, this locating algorithm has a number of shortcomings, which prompted us to create our own algorithm based on Meridian's closest node search [139].

A problem of mOverlay is its topology. Because the groups choose neighbors from their close proximity the logical links between the groups are very short. This affects the performance of the locating algorithm, since the algorithm follows the topology and thus can only make small steps towards the target node. If the target node is far away, taking bigger steps would be more efficient. Another problem is that mOverlay's topology is prone to so-called net-splits.

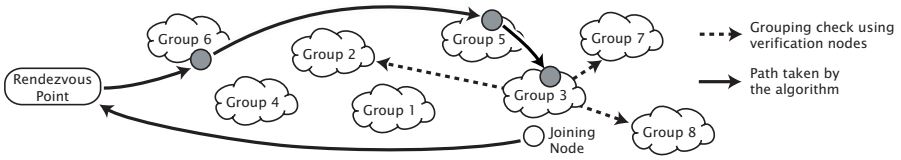


Figure 5.2: Alternative algorithm using Meridian's closest node search and mOverlay's grouping criterion

In order to overcome these problems we have defined an alternative group locating algorithm based on both mOverlay and Meridian (a description of Meridian can be found in Section 3.3). mOverlay groups are constructed based on the following *grouping criterion* [145]:

> When the distance between a new host Q and group A's neighbor groups is the same as the distance between group A and group A's neighbor groups, then host Q should belong to group A.

We take the group concept from mOverlay but change the overlay structure. The groups no longer have direct neighbors. Instead, each group leader joins a network of Meridian nodes and uses Meridian's node cache as neighbor table. When a new node wants to join, it goes through the following procedure:

1. The new node locates a boot node (i.e. a group leader) by sending a request to the rendezvous point.

2. It then asks this boot node to start a Meridian closest node search with itself as target. The search returns the address of the closest group leader to the joining node.

3. At this point, the new node checks the grouping criterion to find out whether or not to join this group. If the criterion is met the new node joins the group. Otherwise, it creates a new group and becomes a Meridian node itself.

The algorithm is illustrated in Figure 5.2.

For checking the grouping criterion we use Meridian's node cache instead of the direct neighbors used in mOverlay's algorithm. The group leader found by the closest node search selects a randomly chosen set of *verification nodes* from its node table and creates a list of addresses and latencies to these nodes. The new node receives this list and in turn measures its latency to each of

the verification nodes. This provides us with two comparable sets. Because mOverlay's grouping criterion is formulated in general terms we need to specify exactly when two distances can be considered "the same." We say distances x and y are the same if

$$\begin{array}{rlll} & x \geq y & \wedge & (1-g) \cdot x \leq y \\ \vee & x < y & \wedge & (1-g) \cdot y \leq x \end{array} \tag{5.2}$$

for a *grouping threshold* $g \in [0, 1)$. The test checks the relative difference between two distances. For example, with a grouping threshold of 0.05 we consider two distances the same if they are within $\pm 5\%$ of each other. A new node joins a group if test (5.2) succeeds for every verification node.

We believe that this combined approach to grouping nodes solves the problems discussed above. Meridian's closest node search makes the search more efficient. The approach is also less prone to net-splits than mOverlay because Meridian nodes maintain a more diverse set of peer nodes. The loose overlay structure also makes the system more resilient to node failures. A possible drawback of our algorithm is that it only checks the grouping criterion for a single group, which bears the danger that the algorithm might skip over the optimal one. Fortunately, the results in Section 5.7.2 suggest that this kind of error is rare.

5.4 Identifying Remote Clusters

5.4.1 Background

In order to make storing and predicting distances to remote nodes more efficient we have developed a method to detect whether two given remote nodes can be clustered based on their proximity to each other. A fundamental restriction we followed is that we cannot expect the remote nodes to run any non-standard software. Consequently, it becomes impossible to ask one of the nodes to measure its distance to the other. The reason for this restriction is that if we do not require any special functionality on the remote nodes we can find clusters of nodes that are not members of the overlay network. Thus, we can use a clustering approach even if we deploy the proposed service only at a single site. In this section we describe a method to identify a possible neighborship of two remote nodes, based only on time series of distance measurements to these nodes made from a single observation point. This method can be used for both, measurements of round-trip time and measurements of available bandwidth (and possibly others we did not consider).

In order to detect neighborships between remote nodes we define distance difference functions $\delta_o(n, m)$, which take values close to 0 if $d_t(o, n) - d_t(o, m) \approx 0$ (the positive case), and larger values otherwise. We require these functions to be commutative, i.e. $\delta_o(m, n) = \delta_o(n, m)$.

With a single observation point a single measurement to two endpoints is not sufficient for cluster detection. However, similarity in several successive measurements may indicate a neighborship. We therefore use time series of measurements for distance difference estimation. From the field of network tomography [20, 28] we know that time series of end-to-end delay measurements can in fact indicate whether two nodes are close to each other in the network

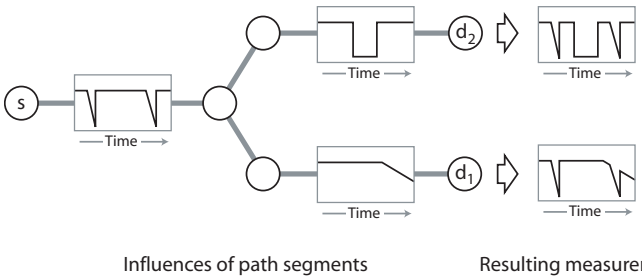


Figure 5.3: Impact of common path segments on end-to-end measurements

topology. This can be used to detect the routing tree between a measuring node and a group of peer nodes [19]. The basic idea is illustrated in Fig. 5.3. The three boxes on the left show the evolution of a given link property (e.g. queuing delay) over time. The two boxes on the right show the impact on the end-to-end measurements. Note that the influence of the common link (the two negative peaks) is observed by both peers, whereas the influence of the other links can only be observed by one peer each. Such similarities can be used to estimate the length of the common path segment. The closer the peers, the more similarities can be observed. In the following we define a measurement-based cluster identification method that uses a different approach than network tomography but is based on the same properties of end-to-end distance measurements.

Time series methods require uniform intervals between observations. However, measurements often come in irregular intervals. They can be described as tuples (t, v, p) of a value v observed at time t for peer p. We define a common base time T_{min} and a time step Δt. The step size is given by the minimum time interval between two consecutive observations of the same peer, i.e. $\Delta t = \min_{p \in P} \left(\min_i (t_{p,i+1} - t_{p,i}) \right)$ where P is the set of all peers. Then, we compute the normalized time series by linear interpolation of the input series at the instants $T_{min} + i \cdot \Delta t$.

Measurements we have performed on the Internet indicate that the standard deviation of available bandwidth grows approximately linear with the mean. We transform this multiplicative relationship between deviation and underlying trend into an additive one by applying the natural logarithm to every element of the normalized time series. Note that this only applies to measurements of available bandwidth.

5.4.2 Distance Difference Calculation

In order to detect whether two measurement targets are neighbors (i.e. belong to the same cluster) we compute distance differences between their respective time series. Distance difference values close to 0 indicate a neighborship. We combine two separate distance difference functions to make the cluster identification process more robust. These distance difference functions are applied to both, time series of round-trip time and available bandwidth, although with different parameters to account for the different characteristics.

If two remote endpoints belong to the same cluster, the distance measured to one endpoint should be a good estimate for the distance to the other. Ac-

cordingly, the first distance difference function identifies pairs of time series x, y where this is the case. We consider y_t to be a good estimate of x_t if y_t lies within an n-percent band around x_t. We detect violations of this rule using the following function

$$o_b(x_t, y_t) = \begin{cases} 0, & (1-b)x_t \leq y_t \leq x_t/(1-b) \\ 1, & otherwise \end{cases} \tag{5.3}$$

Parameter b denotes the size of the band. E.g., $b = 0.02$ denotes a 2% band. We chose the definition of o_b from (5.3) to ensure that $o_b(x_t, y_t) = o_b(y_t, x_t)$. Based on o_b we define the distance difference function O_b, which calculates the ratio of pairs x_t, y_t outside a band of size b of each other.

$$O_b(x, y) = \frac{\sum_{t=0}^{N-1} o_b(x_t, y_t)}{N} \tag{5.4}$$

where N is the length of the times series. A value of $O_b(x, y) = 0$ means that 100% of both time series are inside each others band. Unfortunately, O_b is sensitive to outliers. In order to make the approach more robust to outliers we introduce a threshold parameter $t_O \in [0, 1]$. The criterion for neighbor detection thus becomes $O_b(x, y) \leq t_O$.

The O_b distance difference function in (5.4) detects similar measurement values, but it does not detect systematic differences between time series. Accordingly, we define a second function to check if the time series are unbiased estimators of each other. The relative bias between two time series x and y with mean values $\overline{x}$ and $\overline{y}$ is calculated with

$$B(x, y) = \begin{cases} 1 - \overline{x}/\overline{y}, & x \leq y \\ 1 - \overline{y}/\overline{x}, & x > y \end{cases} \tag{5.5}$$

Again, we have chosen the definition such that $B(x, y) = B(y, x)$. Using the threshold parameter t_B the second criterion becomes $B(x, y) \leq t_B$. This definition of B is closely related to the parameter b of O_b in (5.4): If $O_b(x, y) = 0$ for time series x and y, it follows that $B(x, y) \leq b$. Accordingly, values for threshold t_B should be chosen in the range $[0, t_O]$.

We combine both criteria for cluster identification. Two endpoints are considered neighbors if they satisfy $O_b(x, y) \leq t_O \wedge B(x, y) \leq t_B$. The algorithm therefore depends on the three parameters b, t_O, and t_B. We will investigate the influence of both functions in Section 5.7.2.

We cannot give definite values for the parameters because we do not have any *a priori* knowledge about the statistical error in the input time series. However, the parameters can be approximated by applying cluster identification to a set of known reference clusters. The parameters should be chosen such as to maximize the number of detected neighborships while keeping the number of false positives close to zero. In the following section we investigate the properties of the approach further using a simple statistical model.

5.4.3 The Clustering Algorithm

Based on the distance difference functions from above we can now construct clusters of remote hosts. We use the well-known hierarchical clustering method

[57] for this task. This algorithm requires a measure of distance between two clusters. In our case we use the following *clustering factor* for clusters $C = \{x_1, \ldots, x_n\}$ and $D = \{y_1, \ldots, y_n\}$:

$$G(C, D) = \frac{1}{\overline{C} \cdot \overline{D}} \sum_{i=1}^{n} \sum_{j=1}^{m} G_{host}(x_i, y_j) \tag{5.6}$$

where

$$G_{host}(x, y) = \begin{cases} O_b(x, y), & O_b(x, y) \leq t_O \wedge B(x, y) \leq t_B \\ \infty, & otherwise. \end{cases} \tag{5.7}$$

In other words, the clustering factor between two clusters is the mean out-of-band ratio between its members, or infinity if there is at least one combination of hosts x_i and y_i that cannot be neighbors (i.e., that cannot belong to the same cluster according to the neighbor detection algorithm).

The algorithm works as follows: We start with a set of hosts $h_1, \ldots, h_n$. For each we create a cluster $\{h_i\}$. Then, we find the two clusters C,D with the smallest clustering factor and combine them to a new cluster E, which replaces C and D. We repeat this until either the minimum clustering factor of the set is infinity or there is only one cluster left.

5.4.4 Analysis

The above neighbor identification approach leaves two important points open: The choice of thresholds t_O and t_B, and the number of measurements it takes to reliable accept or reject a possible neighborship. Although the experiments in Section 5.7.2 yield some answers, an analytical model of the approach would help understanding the factors that influence the performance of the approach. In this section we present an analysis of the method based on a simple network model.

Statistical Model

In order to formalize the approach we consider the rather simple statistical model of a measurement scenario shown in Figure 5.4. Consider an observer node and two peer nodes that are possibly neighbors. The paths from the observer to each of the peers share k common links which we call the *trunk*. Then, the paths split into two independent *branches*. We assume that the distance imposed by each link follows a normal distribution and is independent of other links. The notation $X \sim N(\mu, \sigma^2)$ indicates that the random variate X follows a normal distribution with mean μ and variance σ^2. This scenario is a slight oversimplification of reality but it allows for considerably easier modeling of the scenario.

Because of the normality and independence of the single links we can combine the links of the trunk and both branches into three macro-links T, B_1, and B_2 (see Figure 5.5). Just like the single links they follow a normal distribution and are independent of each other.

The time series used for neighbor detection consist of pairs of measurements made simultaneously to both peers. Consequently, the trunk T has no influence on the difference between the measurements. We define $D = B_2 - B_1$ and

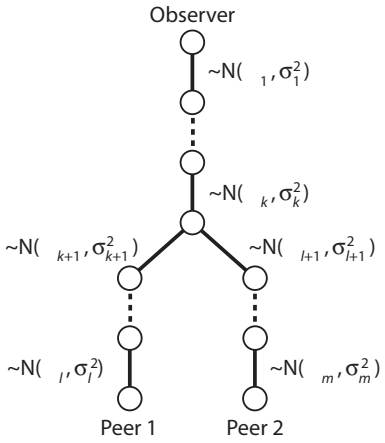


Figure 5.4: Detailed statistical model

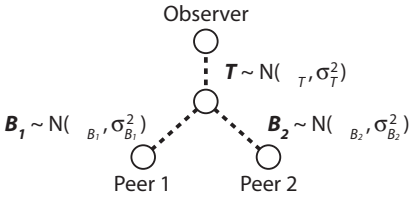


Figure 5.5: Simplified statistical model

note that $D \sim N(\mu_{B_2} - \mu_{B_1}, \sigma^2_{B_2} + \sigma^2_{B_1})$. We also define the random variate $X = T + B_1$ for the distance between the observer and first peer, and $Y = X + D$ for the distance between the observer and second peer.

Analysis of the Out-of-band DDF

Choice of t_O We apply the above model to Equation 5.3, which gives the following rule for in-band measurements

$$\begin{array}{rcl} (1-b)(T+B_1) & \leq & T+B_1+D \\ \wedge \quad T+B_1+D & \leq & (1-b)^{-1}(T+B_1) \end{array} \tag{5.8}$$

The two parts of the expression describe the lower and upper bounds of the test's relative band, respectively. With some transformation we can write the two parts as

$$\begin{array}{rcl} D+b(T+B_1) & \geq & 0 \\ (b-1)D+b(T+B_1) & \geq & 0 \end{array} \tag{5.9}$$

Note that the left parts of the transformed expressions also follow a normal distribution. Since these expressions are derived from the test for in-band measurements, the probability of an out-of-band measurement is the same as the probability that either of these expressions is violated. We can thus compute the theoretical ratio of out-of-band measurements with

$$\begin{array}{rl} O_b = & P\big(D+b(T+B_1)<0\big) \quad + \\ & P\big((b-1)D+b(T+B_1)<0\big) \end{array} \tag{5.10}$$

These probabilities are equal to the CDF of the corresponding normal distributions at 0. The means and variances of these distributions are given below.

For readability we write $\mathcal{U}$ for the term $D + b(T + B_1)$ and $\mathcal{V}$ for the term $(b-1)D + b(T + B_1)$.

$$\begin{aligned} \mu_{\mathcal{U}} &= \mu_D + b(\mu_T + \mu_{B_1}) \\ \sigma^2_{\mathcal{U}} &= \sigma^2_D + b^2(\sigma^2_T + \sigma^2_{B_1}) \\ \mu_{\mathcal{V}} &= (b-1)\mu_D + b(\mu_T + \mu_{B_1}) \\ \sigma^2_{\mathcal{V}} &= (b-1)^2\sigma^2_D + b^2(\sigma^2_T + \sigma^2_{B_1}) \end{aligned} \tag{5.11}$$

Equation (5.10) thus becomes $O_b = P(\mathcal{U} < 0) + P(\mathcal{V} < 0)$ with $\mathcal{U} \sim N(\mu_{\mathcal{U}}, \mu^2_{\mathcal{U}})$ and $\mathcal{V} \sim N(\mu_{\mathcal{V}}, \mu^2_{\mathcal{V}})$.

Using this model we can compute a good choice of threshold t_O for a given scenario: We choose a borderline scenario and a bandwidth b and simply let t_O be the resulting out-of-band ratio O_b. Alternatively, if we wish to use a predefined threshold t_O, we can also determine an adequate bandwidth b given a fixed t_O using numerical approximation.

Required Sample Size A single out-of-band test is not enough to determine a possible neighborship of two nodes. We need several measurements to reliably answer this question. The question is, how many?

The original algorithm detects neighbors if the ratio of out-of-band measurements is smaller or equal than the pre-determined threshold. Given n measurement pairs, the maximum number of out-of-band events is $\lfloor n \cdot t_O \rfloor$. Because the out-of-band test is a Bernoulli experiment (i.e., a yes/no experiment) the number of out-of-bound events is binomially distributed. We consider the borderline case where the probability of an out-of-bound event is equal to t_O. The probability of detection with n samples is given by

$$\sum_{k=0}^{\lfloor n \cdot t_O \rfloor} k \cdot \binom{n}{k} t_O^k \, (1 - t_O)^{n-k} \tag{5.12}$$

Given a threshold t_O and a desired probability of detection we can thus find a suitable number of measurements n.

Analysis of the Relative Bias DDF

Like in the out-of-band case we would like to have a more formal representation of the relative bias test from Equation 5.5. Using the empirical means $\overline{X}$ and $\overline{Y}$ of the random variates X and Y (defined in Section 5.4.4) we can write the DDF as

$$B(\overline{X}, \overline{Y}) = \begin{cases} 1 - \overline{X}/\overline{Y} & \overline{D} \geq 0 \\ 1 - \overline{Y}/\overline{X} & \overline{D} < 0 \end{cases} \tag{5.13}$$

For analysis we distinguish the cases $D \geq 0$ and $D < 0$. In the case $D \geq 0$ we derive the following rules

$$\begin{aligned} t_B &\geq 1 - \overline{X}/(\overline{X} + \overline{D}) \\ t_B(\overline{X} + \overline{D}) &\geq \overline{D} \\ t_B\overline{X} - (1 - t_B)\overline{D} &\geq 0 \end{aligned}$$

The left part of the last inequation follows a normal distribution with

$$\begin{aligned}\mu\big(t_B\overline{X}-(1-t_B)\overline{D}\big) &= t_B\mu_X-(1-t_B)\mu_D\\ \sigma^2\big(t_B\overline{X}-(1-t_B)\overline{D}\big) &= t_B^2\frac{\sigma_X^2}{n}-(1-t_B)^2\frac{\sigma_D^2}{n}\end{aligned}$$

where n is again the sample size.

For the case $D<0$ we obtain similar results

$$\begin{aligned}t_B &\geq 1-(\overline{X}+\overline{D})/\overline{X}\\ t_B\overline{X} &\geq \overline{D}\\ t_B\overline{X}-\overline{D} &\geq 0\end{aligned}$$

Again, the left part of the last inequation is normal, with parameters

$$\begin{aligned}\mu\big(t_B\overline{X}-\overline{D}\big) &= t_B\mu_X-\mu_D\\ \sigma^2\big(t_B\overline{X}-\overline{D}\big) &= t_B^2\frac{\sigma_X^2}{n}-\frac{\sigma_D^2}{n}\end{aligned}$$

Consequently, the relative bias test fails if with probability

$$\begin{aligned}&P(\overline{D}\geq 0)\cdot P(t_B(\overline{T}+\overline{B_1})-(1-t_B)\overline{D}<0)\\ +\;&P(\overline{D}<0)\cdot P(t_B\overline{X}-\overline{D}<0)\end{aligned} \tag{5.14}$$

which can be computed with commonplace numerical tools.

Numerical Examples

Based on this network model we study the behavior of the out-of-band test and the bias test. We consider a trunk length of 95 links and two branches of 5 links each. The links have a mean distance of 1 unit (for example 1 millisecond), with varying standard deviation per link.

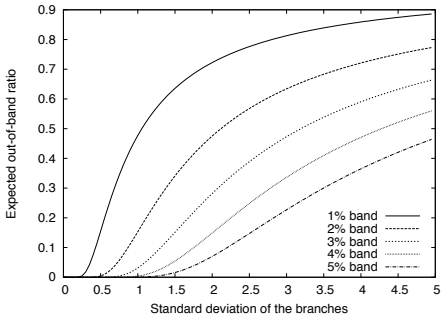
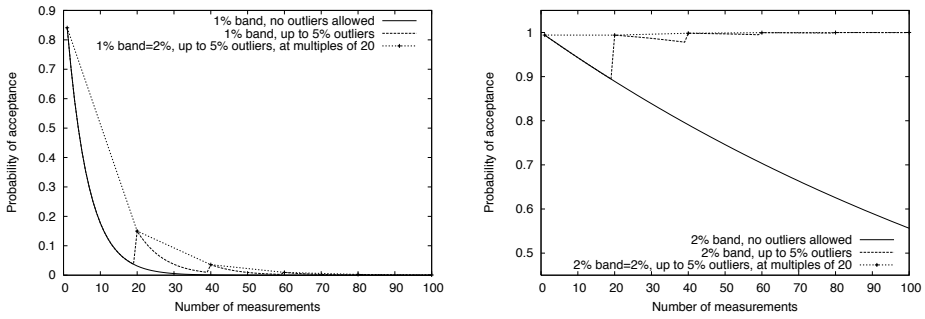


Figure 5.6: Expected out-of-band ratio for several band sizes

Figure 5.6 shows the expected out-of-band ratio for bands of 1–5% and a standard deviation of the branches between 0 and 5 distance units. In this

example, the distance caused by the trunk is zero. We observe that the expected out-of-band ratio never reaches zero unless the standard deviation is zero as well. This means that we must expect to see out-of-band measurements in virtually all situations and should choose the threshold accordingly. A test assuming zero out-of-band ratio will in many cases not detect a neighborship even if the distances to both targets is very similar. Since in a real network there is always the chance of an outlier we have to choose $t_O > 0$ in order to reliably detect neighbors.

In a second example we study the probability to detect a neighborship depending on the number of measurements. Here, each link has a standard deviation of 10% of its mean distance. If we use a 1% band the test should reject a possible neighborship of the targets. With a 2% band it should detect a neighborship. Figure 5.7 shows the probability to detect a neighborship for 1–100 measurements.



(a) 1% band (b) 2% band

Figure 5.7: Acceptance probability of the out-of-band test with 10% standard deviation

Figure 5.7(a) shows the case of a 1% band. If we choose $t_O = 0$, 18 measurements are enough to reject the neighborship with 95% probability. With $t_O = 5\%$ however, we observe "steps" in the probability graph. These are due to the nature of the test. If we observe an outlier in 19 measurements we reject the neighborship because the percentage is greater than 5%. However, if we observe and outlier in 20 measurements, it is within these 5%. Accordingly, when we use a threshold $t_O > 0$ we have to choose the number of measurements accordingly. With $t_O = 5\%$ we use multiples of 20 measurements. The third graph in Figure 5.7(a) shows that in this case 40 measurements are necessary to reject a neighborship with more than 95% probability.

As mentioned above, threshold t_O should never be zero. Figure 5.7(b) illustrates this. With zero threshold the probability to detect a neighborship declines with a rising number of measurements due to the rising probability of at least on outlier. Conversely, with a 5% threshold 20 measurements are enough to detect the neighborship with 99% probability.

A weakness of the out-of-band test is that it cannot detect a small but systematic difference between two targets if there is very little variance in the distance measurements. The bias test compensates for this weakness. In the following example the trunk still consists of 95 links, but the branches now

have 4 and 6 links, respectively. This should be detected by the bias test when checking for a 1% relative bias or greater. The standard deviation on each link in this example varies between 0 and 75% of the link's length.

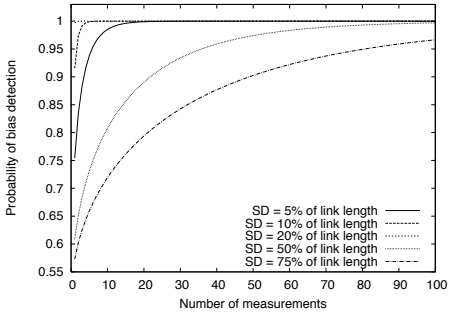


Figure 5.8: Probability to detect a 1% bias for varying standard deviation

From Figure 5.8 we see that the bias can be reliably detected with very few measurements if the standard deviation is small. For very large standard deviation, the number of necessary measurements becomes much greater. Nonetheless, this is not a problem since the bias test is aimed at compensating errors in low variance scenarios.

5.5 The Peer-to-Peer Prediction Architecture

The architecture of our end-to-end distance prediction service is based on a peer-to-peer approach where nodes that are close together in the underlying topology join the same groups to exchange and coordinate measurements. We use a two-layer architecture. On the loosely connected *global layer*, inter-group communication takes place. On the *local layer*, the members of each group are integrated into separate, structured peer-to-peer networks that serve for efficient, distributed storage of measurement data and prediction models (see Figure 5.9).

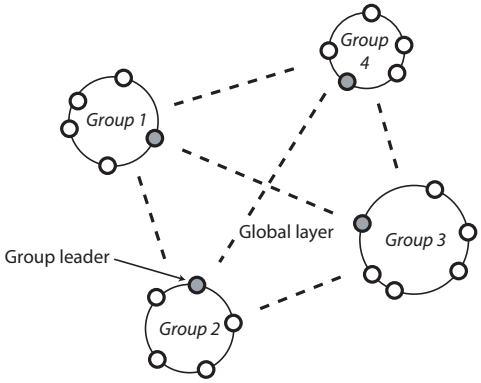


Figure 5.9: The two-layer structure of the service

Groups are largely independent of each other and can function without interaction to other groups, with the exception of the group localization procedure new nodes go through to find a suitable group to join. Each member of a group provides an API to client processes on its end system. Clients can request measurements and predictions through this API. In order to take advantage of the implicit measurements done by normal traffic, applications can also send such measurements back to the service. For example, when a web server has sent a large file to a client it can submit the achieved TCP throughput value to the service. A network traffic monitor would also use the same API to submit measurements. The group maintains a repository of measurements and prediction models distributed among the peers. In order to reduce the number of remote endpoints that the repository has to keep track of, we use clustering. Remote nodes that are not members of the distance prediction service (so-called *non-member* nodes) are handled independently. Remote nodes that belong to the same remote group are treated as a single unit. Both, remote groups and non-member nodes, are further clustered using the algorithm from Section 5.4 (when we use the term *cluster* in the following we generally refer to these). Figure 5.10 illustrate these levels of clustering.

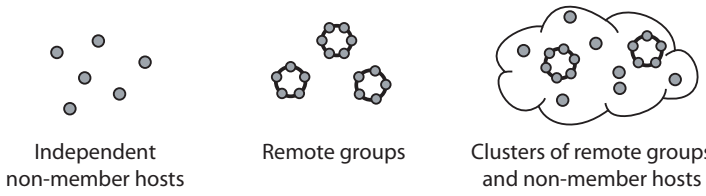


Figure 5.10: The three levels of clustering used in the repository

Even with clustering, the repository would hardly be able to store information about every possible host on the Internet. The approach we take with our distance prediction service is related to the concept of proxy servers. The service's peer-to-peer groups normally correspond to the local networks of single organizations such as enterprises or universities. Since the users of an organization share common interests, their requests concentrate on a relatively small set of popular IP addresses. Therefore, if the service only stores information about the most popular clusters it can still give predictions for the majority of requests. Moreover, since our service relies heavily on measurements gained from monitoring application traffic, popular clusters tend to have more measurements available than unpopular ones, which in turn leads to better predictions.

Based on these observations we manage the size of the repository using a least-recently-used strategy. Each group member defines an upper limit for the amount of data it stores in its local storage (e.g. its harddisk). When this limit is exceeded it removes the least popular clusters from its part of the repository until enough storage space has been freed. The popularity of a cluster is measured by the time elapsed since the last request for it has been received.

5.5.1 The Global Layer

The global layer is an unstructured collection of independent groups without any fixed topology. Both group leaders and normal group members may freely

communicate with each other. However, since there is no topology and no directory of members, communication can only take place if two peers have learned about each other during the normal operation of the network.

The global layer's main purpose is the bootstrapping of new nodes. In order to support bootstrapping, the *group leader* of each group runs a Meridian [139] process. When a new node joins the peer-to-peer network it uses the Meridian closest node search to locate its nearest group. As soon as it finds the group it uses the neighbor table of the group leader's Meridian process to check the grouping criterion. If the check is successful the joining node contacts the group leader to join the group. Otherwise, it creates its own group, declares itself leader, and starts a Meridian process to allow other nodes to find the new group. A more detailed description of this procedure can be found in Section 5.3.

The second purpose of the global layer is the identification of remote groups. When a node receives a request referring to an IP address that is not known in its local repository, it will try to send a message to that address on a well-known port, asking for the ID of the remote node's group. If the destination node is a member of the system it will answer, allowing us to look for existing information about that group ID in the local repository. If the remote node is a non-member the message will time out after a short delay, and the request fails.

5.5.2 The Local Layer

On the local layer nodes are organized into groups, each of which is made up of a Pastry [109] ring. Using its Pastry ring, each group maintains a distributed repository of identified clusters, measurement data, and prediction models. The members of a group are controlled by its group leader. When a new node joins the group it communicates with the leader to become part of the ring. This is done by computing a Pastry ID that uniquely identifies the new node, and which determines the node's position in the ring. IDs are computed using an MD5 hash on a string containing the node's IP address. A given IP address always results in the same ID. There are four types of IDs:

Node IDs	designate a specific member node of the local group and determine the portion of the group's repository the node is responsible for.
Cluster IDs	designate identified clusters of non-member hosts and remote groups.
Group IDs	designate known remote groups and are used to look up information about them.
Host IDs	designate known remote hosts including both, non-member nodes and nodes belonging to a remote group. The IDs are unique for a given IP address.

Node, cluster, and host ID need only be unique within the repository. Group IDs however are globally unique since they are used for identifying different groups on the global layer.

Information in the repository is always stored by cluster ID. The Pastry node with the numerically closest node ID is responsible for storing the data.

If the group learns of a new remote host or group that cannot be assigned to a cluster, a new cluster is created that contains only this host or group. Later, when enough measurement data is available to run the clustering algorithm the new cluster may be merged with another.

The Pastry ring maps each host and group ID to its assigned cluster ID. Accordingly, a lookup for a given ID first resolves the assigned cluster ID and then locates the Pastry node responsible for storing the relevant data. When a node wants to retrieve the latest measurement data for a given IP address, it computes the host ID of that address and performs a lookup on the Pastry ring. If the ID is known it receives the ID of the host's cluster. The requesting node then sends a message to the node responsible for that cluster ID to retrieve the desired data. Finally, once the message reaches the responsible node, it returns the data (or an error message if the data is not available) directly to the requester. Figure 5.11 illustrates this procedure.

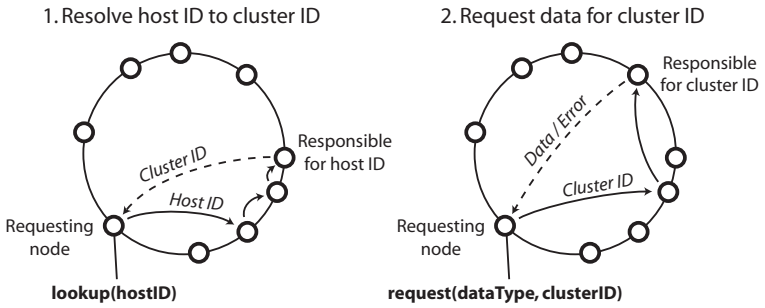


Figure 5.11: A node retrieves information about a known IP address

If the local repository has no information about the remote host the initial lookup fails. However, if the remote host is a member of the distance prediction service the repository might contain information about the remote host's group. Accordingly, the requesting node will send a message to the remote host on a well-known port, hoping for an answer. If successful, the requesting node receives the (globally unique) ID of the remote host's group. Using this group ID we can then perform a second lookup similar to the one for the host ID. In case the group ID is known it will resolve to a cluster ID, which can then be used as normal. Otherwise, the system returns an error message to the client.

Client applications will quite frequently send queries about unknown host IDs to the local group. Consequently, we have to add a new entry to the repository. We distinguish three cases:

1. The repository contains no information about the host ID, and we were not able to obtain a group ID from the remote host.

2. The repository contains no information about the host ID. The remote host returned the globally unique ID of its group, which is not known either.

3. The repository contains information about remote host's group, but not about the remote host itself.

The first case we have to insert a record for a non-member node into the repository. We start by inserting a "busy" entry for this host ID into the Pastry ring.

This is to eliminate possible conflicts when another group member attempts the same operation simultaneously. Since the new host cannot be assigned to any existing cluster we generate a new cluster ID and tell the group member responsible for that ID to create a new cluster data structure containing only the new host. Then, we change the host's Pastry entry from "busy" to the new cluster ID. Such small clusters are inefficient, so we consequently try to merge it with another cluster as soon as possible. Accordingly, we schedule a series of round-trip time measurements to the remote host in order to obtain the data necessary to run the clustering algorithm.

In the second case we have to create an entry for a new group in the repository. The procedure is analogous to the first case. We create a new cluster and insert an entry resolving the group ID to its cluster ID into the Pastry ring. Then, we schedule measurements to the new host since it is the only known member of the new group. However, in contrast to the first case we also insert an entry for the new host ID into the Pastry ring. This entry resolves directly to the new cluster ID rather than to the group ID.

The third case is the most simple since the new host's group is already known in the repository. Accordingly, we can simply add a Pastry entry that links the new host ID to the same cluster as the group ID resolves to.

5.5.3 Repository Maintenance

Every group member stores and manages all clusters with IDs that are in its ID range on the Pastry ring. This includes storing measurements made to hosts in these clusters, computing distance prediction models, and realigning clusters where necessary.

Measurements for a cluster are stored in a special directory on the node's local storage (e.g. its harddisk). There are subdirectories for each cluster and every host or group contained in these clusters. Subdirectories for hosts and groups contain the measurements specific to them. Additionally, the measurements of all hosts and groups of a cluster are combined and stored in the cluster's subdirectory. The node keeps track of the amount of data for each entity and the time the measurements were made. To save space, old data may be discarded. If a node does not receive any requests for a cluster it may also entirely remove it from the system. The reason for keeping the data of hosts and groups in separate directories is that clusters may be merged and split. In this case the combined data of the old cluster is obsolete and must be recalculated for the new clusters. The data is also needed as input for the clustering algorithm used in merge and split operations.

Once there is enough measurement data of a given type for a cluster, the node creates a prediction model for it and stores it in its local storage along with the data. Prediction models are updated regularly when the node receives new measurements in order to improve them and fit them to recent data. Section 5.5.5 describes the prediction method we use.

The clustering of hosts and groups may be flawed when based on too little or inaccurate measurements. Bad clustering may also be due to routing changes in the network in the case they affect the distances from the local group to the cluster's remote hosts and groups. Accordingly, the repository may realign clusters if it detects the need to do so. This operation, called *cluster splitting*, may also be done on a regular basis. The process takes place on the node

responsible for the cluster. It constructs time series for each of the cluster's hosts and groups from the measurements in their respective subdirectories and applies the clustering algorithm from Section 5.4. If the algorithm only identifies a single cluster the node continues as before. If it identifies two or more clusters, the node retains the biggest one and creates so-called *cluster packages* for the others, which it sends to other members of the group according the the clusters' IDs. There, the new clusters will be stored locally.

Cluster merging, the converse operation from cluster splitting, becomes necessary when there are too many small clusters in the repository. Unlike cluster splitting, merging often concerns clusters stored on separate group members. Accordingly, the peers have to detect possibilities to merge clusters and coordinate the merge operations.

From time to time each group member creates short time series describing the most recent round-trip time data for all of its clusters and sends them to all other group members using a Pastry broadcast message. In addition to the time series these messages also contain the sizes of the respective clusters. The receivers try to match these time series with the clusters they store, using the neighborship criteria from Section 5.4. If a receiver detects a possible match it will contact the sender to initiate cluster merging. The merging algorithm is always executed on the member storing the bigger cluster (in terms of the number of nodes and groups contained in the cluster). Accordingly, the receiver uses the push or pull variant of cluster merging depending on the sizes of both clusters. The actual merging is done by running the clustering algorithm on the combined set of members of both clusters. This operation may result in multiple clusters, which are handled the same as in cluster splitting. The largest cluster remains on the node and the others are sent to other group members according to their cluster ID.

A problem with this broadcasting of time series is that the more clusters are stored in the repository, the more bandwidth is used for the broadcasts. We solve this problem using two mechanisms. On the one hand, the rate at which information about a cluster gets broadcasted is made smaller the longer the cluster remains in the repository. Thus, new clusters quickly find existing clusters to merge with and old ones do not use bandwidth unnecessarily. As a result, the amount of traffic does not depend on the total number of clusters in the system but rather on the number of number of newly created clusters. On the other hand, the broadcast rate for each cluster is reduced when the number of group members grows. This keeps the bandwidth per node used for this task constant. Since a group's size is restricted by the network topology, the resulting rate of cluster information broadcasts never reaches zero.

Our distance prediction service is based on the assumption that a large part of the measurements comes from monitoring application traffic. Nevertheless, group members may be configured to schedule active measurements for clusters that do not receive a sufficient amount of data from client applications. In this case they run timers for each cluster, counting the time since the last measurement for that cluster has been received. When a timer exceeds a threshold, the node randomly chooses a target IP address from the cluster and schedules a round-trip time measurement (other measurements may be too costly to be scheduled automatically). This ensures a minimal amount of measurements necessary for the realignment of clusters.

5.5.4 Resilience to Node Failures

An important factor for a peer-to-peer architecture is its resilience to node failure. A group members may fail for various reasons such as faulty software, loss of network connectivity, or simple because a computer has been turned off. If the system does not take any precautions, the data stored on the node is lost. Pastry addresses this problem by replicating the keys managed by a node on nearby nodes on the ring. Thus, when a node fails its data can in most cases be recovered. We exploit this feature of Pastry to distribute backups of the clusters stored by a node. Each node creates backup copies of its clusters and stores them on nearby nodes according to Pastry's key replication strategy. When a node receives a request for a cluster it is not responsible for but of which it has a backup, it assumes that the original responsible for that cluster must have failed. Accordingly, it extracts the data from the backup file and assumes responsibility for the cluster. Since backups are only made once in a while we will probably lose the most recent measurements and computations made for the cluster. However, given that node failures are relatively rare, the error resulting from this small loss of data should be small.

A special case occurs if the group leader fails. Since it is only needed to help new nodes find their place in the overlay network, a failed group leader does not interrupt the normal operation of the group. Nevertheless, new nodes could no longer find and join the group. In order to recover from leader failures we use the scheme proposed for mOverlay groups in [145]. The group leader regularly replicates its state to one or more backup leaders. The backup leaders have a fixed sequence. If the first backup leader does not receive any message from the group leader for a certain time it activates the stored backup state, starts a new Meridian process to enable new nodes to find the group, and declares itself leader. It also assigns a new backup leader to replace itself. If both the group leader and its first backup leader fail simultaneously, the second backup leader takes over, et cetera. Hence, we can adjust the group's resilience to leader failures by choosing the number of backup leaders.

5.5.5 Predictions

Once there is sufficient measurement data of a specific type for a cluster, a statistical prediction model for this type will be computed. The computation takes place on the node responsible for the cluster, and the model is stored in the same location. Later, when a client requests a prediction for the cluster, the node retrieves the model from the harddisk together with the latest measurement data and returns both in a *predictor* object. The reason for not just returning a simple predicted value is that the client might be interested in other aspects of the prediction, such as the estimated error.

We use auto regressive (AR) models (described in [13]) for predicting all available kinds of measurements. There are other models that are better suited for specific kinds of measurements. However, AR models are relatively cheap to compute and provide acceptable predictions regardless of the type of measurement. Nevertheless, more elaborate models could be integrated into the architecture.

An AR model considers a given time series of measurements as generated by a random process

$$x_t = \phi_1 x_{t-1} + \ldots + \phi_p x_{t-p} + \varepsilon_t \quad . \tag{5.15}$$

i.e. the current value x_t is a linear combination of the preceding p values, plus a normal error term. The time step between x_t and x_{t+1} is chosen depending on the type of measurement. We call p the *order* of the model. The above formulation also assumes that the mean $\overline{x}$ of the series is zero, which can be easily achieved by subtracting the estimated mean $\hat{x}$ from the values. An AR model can thus be described by its parameters $\phi_{1\ldots p}$ and the mean $\overline{x}$ of the series. We can fit an AR model to a series of measurements using the Yule-Walker equations [143], which reduce the problem to solving a linear system of equations based on the autocorrelation function of the measurement series (a brief explanation of the equations is given in Appendix A.2). Given an AR model, we can then calculate a prediction simply by replacing the x_{t-i} in (5.15) with the appropriate values from the measurement series, and assuming that $\varepsilon_t = 0$.

A predictor object for an AR model only contains $2 \cdot p + 2$ values: p parameters, the p latest values from the measurement series, the estimated mean value $\hat{x}$ of the series, and the estimated error of prediction. Using an AR predictor object a client can calculate predictions for one or more time steps into the future, each step with an estimated error of prediction.

Models are normally updated on a regular basis using a pre-configured order p. However, if a cluster has only recently been created we probably do not have sufficient measurement data to compute an AR model of order p. Depending on the available amount of data the node will therefore select a lower order q. In this case the node also reduces the normal updating interval for the model in order to quickly adjust the model when new measurement data arrives. The accelerated updating interval is given by $\Delta t \cdot \alpha^{-(p-q)}$, where Δt is the normal updating interval and α is an acceleration factor.

5.6 Simulations

In this section we evaluate the algorithms for identifying groups and clusters. The evaluation of the algorithms was based on publicly available measurement data also from PlanetLab [130].

5.6.1 Identification of Local Groups

Simulation Approach

In order to compare the performance of mOverlay's locating algorithm to our alternative algorithm we have implemented simulators for the two approaches. Both simulators are based on a black box network model given by a matrix of the end-to-end latencies between each pair of endpoints in the simulation. For our experiments we use a matrix derived from all-sites ping data measured on PlanetLab [130]. In both simulators the nodes join the overlay network one after the other, in pseudo-random order (given by the seed value). For each node we record the time that expires until it joins a group or creates its own

group. When the simulation ends we examine the resulting groups according to several criteria, which we discuss in Section 5.6.1.

mOverlay We simulate mOverlay with a simple message-based approach where each message fits into a single packet and the message processing at a node does not take any time. Thus, a request-response message exchange takes exactly one round-trip time to complete, which is a lower bound for any real implementation of the framework. Furthermore, we skip mOverlay's initial request to the rendezvous point because the performance of this step depends heavily on the implementation of the mechanism (e.g. a well-known address, a DNS-based approach, etc.) and possibly on the placement of the the rendezvous point.

In the simulator, the locating processes of a joining node run in parallel and stop when one of them finds a group that meets the grouping criterion. A locating process also stops if its next hop would be a group it has already visited. If none of the locating processes are successful, the joining node gives up and creates a new group. Locating processes keep a list of visited groups. When a new group is created its neighbors are selected from the lists of all its locating processes. The first two joining nodes are special cases. They automatically create new groups because the grouping criterion cannot be evaluated without further nodes. As mentioned in Section 5.3, mOverlay does not define how to test if two distances are the same. However, we need to test this to check the grouping criterion. We have used test (5.2) from Section 5.3 also for the mOverlay simulation because it is a natural choice.

Meridian In contrast to the mOverlay simulator, where we implemented all necessary messages, we did not implement the Meridian approach ourselves. Instead, we have used the Meridian C++ implementation available at [138]. We have written wrapper code to redirect any messages to a simulation back-end instead of the network, and we have changed Meridian's time-keeping code to use the simulation time instead of the system clock. Each Meridian node is now a C++ object in the simulator rather than a physical node on the network. When it sends a packet the simulator determines the appropriate transmission latency using the underlying network model and schedules the packet arrival at the destination node accordingly. The wrapper objects also evaluate the grouping criterion at the end of a joining procedure and create a new group if necessary.

The simulation back-end is event-based. There are three kinds of events: one for inserting a new node into the scenario, one for triggering Meridian's periodic gossip protocol, and one for packet arrivals at a Meridian node. We start the simulation by scheduling node join events every seven seconds (which corresponds to Meridian's default gossip interval). When a node joins it starts by sending a closest node query to a Meridian node. This search is handled entirely by the original code. When the query returns, the joining node contacts the identified closest node to retrieve a list of verification nodes, which the wrapper code extracts from the Meridian object's latency cache. In the simulator we use a maximum of five verification nodes.

Simulation Results

Simulation Scenario For the simulations we have used a matrix of round-trip times between 77 PlanetLab nodes, based on all-sites ping data from PlanetLab [130]. The simulator estimates the one-way delay between two endpoints by dividing the appropriate round-trip time by two. At the time of writing, 694 machines hosted by 335 sites were part of PlanetLab. This means that each site hosts only slightly more than two machines on average. Consequently, we can expect to find relatively small groups in our scenario, especially since the 77 nodes in the network model were randomly selected from the available PlanetLab nodes. For each node pair we have also acquired a time series of round-trip times, which we use for evaluation. The time series contain round-trip time measurements every 15 minutes during one day. Simulations have been run with different values for various parameters. Furthermore, each set of parameters was simulated using 100 different seeds, which we obtained from from random.org [102].

Evaluation Criteria We get the joining delays for every node, and the identified groups from a simulator run. While the comparison of the joining delays is straightforward, quantifying the quality of the identified groups is not. Grouping can exhibit two kinds of errors, false positives and false negatives. A node joining a group when it should not is considered a false positive and increases the error of grouping. A false negative occurs when a node erroneously does not join a group and creates a new one instead. This results in too many groups and impairs the efficiency and scalability of the overlay network. Unfortunately, due to the black box nature of our network model, we cannot say a priori whether a node should join a group or not. Nevertheless, we can define three criteria for the quality of the identified groups.

- First, the members of a group should be close to each other. Accordingly, we compute the mean round-trip time between members of the same group. Groups with only one node are ignored in this case.

- Second, bigger groups are preferable because they reduce the complexity of the overlay network. We use the average number of nodes per group as the second criterion.

- The third criterion stems from the use of the identified groups as a basis for a distance estimation service. One important assumption in mOverlay is that if two nodes A and B are in the same group, the distances $\overline{AC}$ and $\overline{BC}$ to a node C outside the group are virtually the same. This property must also hold over time. Otherwise, we would have to reorganize the groups constantly. We define the third criterion accordingly: If A and B are in the same group, $\overline{AC}_t$ should be a good prediction for $\overline{BC}_t$, where t is the time of measurement. We verify this using the out-of-band test from Section 5.4 applied to the time series of round-trip times between the two nodes. Two measurements $\overline{AC}_t$ and $\overline{BC}_t$ are out-of-band of each other if

$$\begin{array}{rlll} & \overline{AC}_t \geq \overline{BC}_t & \wedge & (1-b)\cdot\overline{AC}_t > \overline{BC}_t \\ \vee & \overline{AC}_t < \overline{BC}_t & \wedge & (1-b)\cdot\overline{BC}_t > \overline{AC}_t \end{array} \tag{5.16}$$

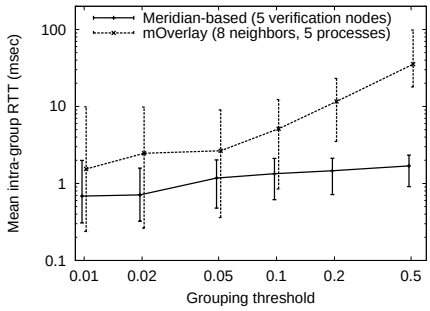


Figure 5.12: Mean intra group distance for several grouping thresholds

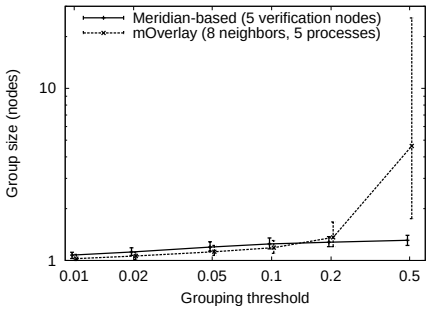


Figure 5.13: Avg. nodes per identified group for several grouping thresholds

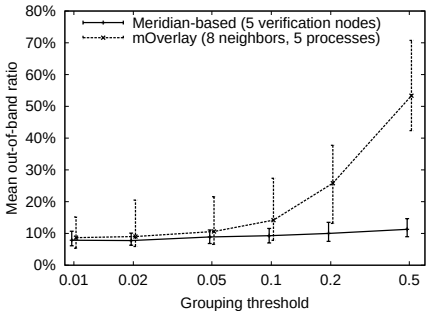


Figure 5.14: Mean out-of-band ratio using a 10% band, for several grouping thresholds

for a relative bandwidth $b \in [0, 1)$. The *out-of-band ratio* between two nodes is the ratio of out-of-band measurements in the respective time series. In this section we use a bandwidth of 10% ($b = 0.1$).

The graphs in the remainder of this section use a dot-and-whisker format showing the mean with a 95% confidence interval, obtained by running the simulation with 100 different seeds. We have also slightly staggered the graphs along the horizontal axis to improve readability.

Quality of the Groups As a first comparison we look at the quality of the groups identified by mOverlay and our alternative approach. For both we use parameters that we have found to produce near optimal results. We set the maximum number of neighbors for mOverlay groups to eight and the number of parallel locating processes to five. For the Meridian-based approach we set the maximum number of verification nodes to five.

Figure 5.12 shows the mean round-trip times between group members for the grouping thresholds 1%, 2%, 5%, 10%, 20%, and 50% (using a logarithmic scale for better readability). For both approaches a lower threshold also leads to smaller distances between group members. The effect is much bigger for mOverlay because the grouping threshold affects every iteration of the locating process,

while the Meridian-based locating algorithm only uses the grouping threshold for its final step. Nevertheless, the round-trip times between group members of mOverlay are always bigger on the average than those of the Meridian-based approach. Moreover, the confidence intervals for mOverlay are bigger. We conclude that the Meridian-based approach performs better than mOverlay with respect to the first criterion.

The second aspect we examine is the average number of nodes per group. Figure 5.13 shows the group size for the same grouping thresholds as Figure 5.12. The groups identified by the alternate approach are bigger for grouping thresholds up to 10%. In contrast, mOverlay identifies much bigger groups with grouping thresholds above 10%, but this comes at the price of much greater round-trip times between group members. As expected, group sizes are rather small because of the wide distribution of the nodes.

If the identified groups shall be used as a basis for a distance estimation service they must also have a low out-of-band ratio. We look at this aspect using again the same parameters for grouping threshold and a 10% band for the out-of-band test. The results can be seen in Figure 5.14. The Meridian-based approach has shows a smaller out-of-band ratio than mOverlay for all grouping thresholds, and it shows less variance. Again, mOverlay shows high sensitivity towards the grouping threshold while the out-of-band ratio of the Meridian-based approach only increases slightly with growing grouping threshold.

Joining Delay In addition to a good quality of the identified groups it is also desirable to find the groups in the shortest time possible. We compare the two approaches using the same parameters as in Section 5.6.1. Figure 5.15 shows the joining delay per node for several grouping thresholds. Again, mOverlay proves to be much more sensitive towards the grouping threshold than the Meridian-based approach. Moreover, unless the grouping threshold is extremely high the alternate algorithm finds the local group much faster than mOverlay.

The joining delay of mOverlay nodes is not only sensitive to the choice of grouping threshold. Figure 5.16 shows the influence of the maximum number of neighbors per group and the number of parallel locating processes. Here we used a grouping threshold of 5% and a maximum of 2–10 neighbors per group. Furthermore, the three graphs show the effect of using 1, 5, or 10 parallel locating processes. We observe that a lower maximum of neighbors per group and fewer locating processes running in parallel cause a significant reduction in joining delay. Furthermore, the increase in joining delay appears to become smaller the more parallel locating processes we employ. However, regardless of the parameters the confidence interval is always rather large.

It appears that mOverlay can match the speed of the alternate approach if we reduce the number of parallel locating processes and the maximum number of neighbors per group. Nevertheless, the effect on the quality of the groups is severe as Figure 5.17 shows. Less than ca. 6 neighbors per group cause a significant increase in the out-of-band ratio. Using only one locating process also has a noticeable negative effect. On the other hand, the benefit from using more locating processes rapidly declines. The difference between five and ten parallel locating processes is mainly the size of the respective confidence intervals.

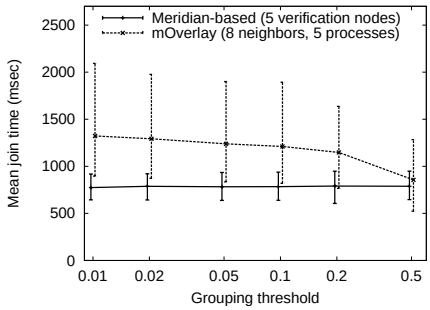


Figure 5.15: Mean joining delay per node for several grouping thresholds

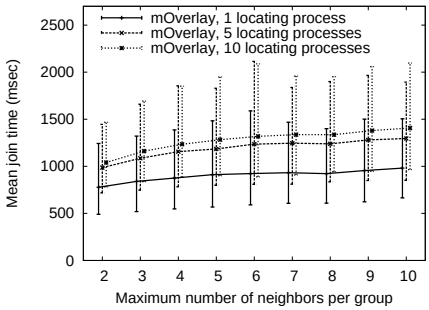


Figure 5.16: Mean joining delay in mOverlay for various parameters

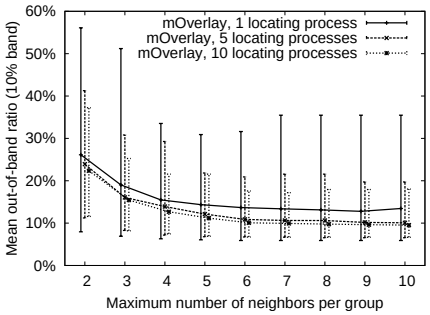


Figure 5.17: Mean out-of-band ratio in mOverlay for various parameters

Figure 5.17 also justifies our choice of parameters for mOverlay. The improvement for more than a maximum of eight neighbors per group and five locating processes is small while the increase in joining delay is still noticeable.

From the simulation results we conclude that the Meridian-based locating algorithm is faster in most cases. It also identifies larger groups, and the nodes inside the groups are closer together than the nodes in mOverlay groups. Moreover, the groups identified with the alternate algorithm also have a smaller out-of-band ratio, which indicates better suitability for a distance estimation service.

5.6.2 Remote Cluster Identification

The remote neighbor identification procedure described in Section 5.4 is important for the scalability of our architecture. The neighbor identification procedure was evaluated in three separate experiments. In the first experiment we investigate the impact of the three parameters b, t_O, and t_B based on a large set of averaged round-trip time measurements, and we show that the approach is able to reliably identify clusters. The second experiment demonstrates that the approach also works with non-averaged round-trip time measurements. In the third experiment we use measurements of available bandwidth to identify clusters.

Cluster Identification with Averaged Round-Trip Times

In the first experiment we have used data from PlanetLab [130] consisting of time series of round-trip time measured every 15 minutes during three days. Each value in the time series represents the mean of 10 RTT probes. The measurements were done between 77 PlanetLab nodes (these were the same nodes used Section 5.6.1). Every endpoint measured the round-trip time to each of the other 76 endpoints, resulting in $77 \cdot 76 = 5852$ time series of round-trip time. The original data from PlanetLab included measurements from more than 77 endpoints, however with gaps. Therefore, we have used the maximum subset of endpoints that provided a full mesh of complete measurements. Cluster identification was done using the measurements from the first 1.5 days. The measurements from the second 1.5 days were used to verify the results.

Since measurements were available from each of the 77 endpoints to all other endpoints we have performed 77 cluster identification procedures, one for each endpoint as an observation point. Each time, we computed the distance difference functions for every pair of the 76 other endpoints. Depending on the thresholds t_O and t_B we then decided whether a given pair of endpoints are neighbors (belong to the same cluster). These computations were repeated with several different values for parameters b, t_O, and t_B. In order to examine the influence of bias detection we have also performed this experiment without using the distance difference function B.

Two criteria were used to verify the results of the experiment. First, if two given endpoints were identified as neighbors from a given observation point, then the second 1.5 days of the measurements should confirm this. The second halves x', y' of the respective time series should still be good estimates of each other and thus satisfy $O_b(x', y') \leq t_O$ for suitable values of b and t_O. Second, the average round-trip time between neighbors should be small compared to the average round-trip times between other pairs in the data set. The full-mesh structure of the available measurement data allowed us to verify this easily. We discuss the results of applying both criteria in the following two sections.

Verification Using the Second Half of the Measurements For every pair of endpoints identified as neighbors we verified whether the second halves of the respective time series show similar behavior. We considered the detection confirmed if the time series' values were within a 5% band of each other 95% of the time. Using the O_b function from (5.4) we can formulate this as $O_{5\%}(x', y') \leq 5\%$.

We have compared the number of confirmed neighbor detections with the number of total detections for several parameter sets. Fig. 5.18 shows the results for several values of b (the size of the relative band in O_b) and of t_O (the maximum ratio of values outside the band). The bias detection parameter t_B was constantly 0.1%. We can see from Fig. 5.18 that 100% of the neighbor detections were confirmed for small values of t_O and b. This shows that the approach is able to correctly determine whether two given endpoints belong to the same cluster. Values of b greater than 3 resulted in smaller ratios of confirmed detections. The choice of parameter t_O also significantly influences the quality of the detections. The greater t_O, the lower the ratio of confirmed detections. Nonetheless, only $t_O = 5\%$ and values of b greater than 3 resulted in less than 90% confirmed detections.

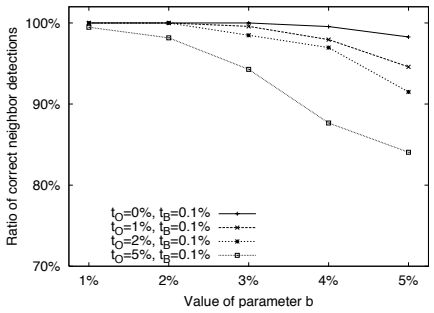


Figure 5.18: Ratio of confirmed neighbor detections with $t_B = 0.1\%$

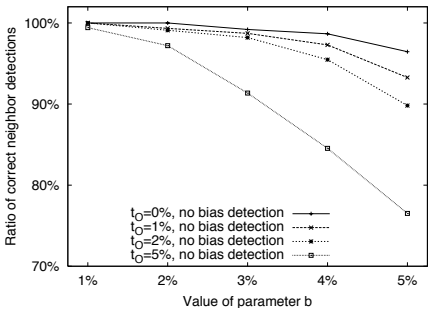


Figure 5.19: Ratio of confirmed neighbor detections without bias detection

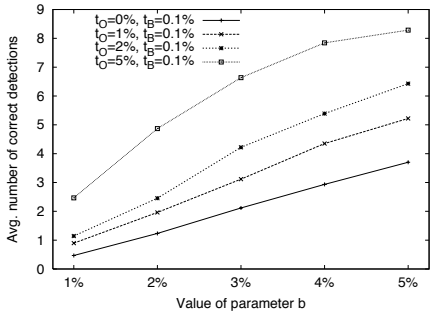


Figure 5.20: Number of confirmed neighbor detections per observation point

In order to evaluate the contribution of bias detection to the cluster identification method we have performed the same experiment using only the out-of-band criterion from (5.4). If bias detection really has a positive influence we should observe a significantly smaller ratio of confirmed neighbor detections. Fig. 5.19 shows the results of this experiment. While the ratio of confirmed neighbor detections only slightly decreases for small values of t_O and b, the effect is significant for greater values. For example, the ratio of confirmed detections with $t_O = 5\%$ and $b = 5\%$ decreases by 7.5%, from 84% with bias detection (Fig. 5.18) to 76.5% without bias detection (Fig. 5.19).

For all parameters, small values lead to better ratios of confirmed neighbor detections. However, there is always a trade-off between the number of confirmed neighbor detections and the number of false negatives, i.e. the number of endpoint pairs that would be confirmed as neighbors but are not detected as such. Fig. 5.20 illustrates this. The average number of confirmed neighbor detections per observation point is very small for $b = 1\%$ and rises considerably with higher values of b. This trade-off must be considered when choosing parameters for cluster identification. The criteria should not be more restrictive than necessary for a given application. Note that the number of identified neighbor pairs per observation point is rather small because PlanetLab [95] nodes are widely dispersed throughout the Internet, with only a few nodes per site. At

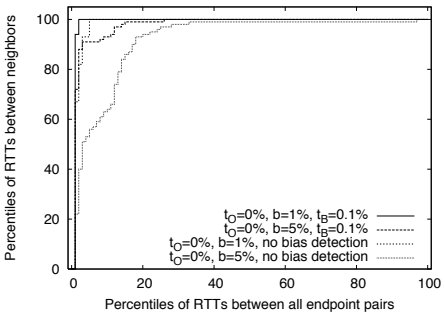


Figure 5.21: Percentile-percentile plot of round-trip times between identified neighbors versus the round-trip times between all endpoint pairs

the time of writing, PlanetLab consisted of 606 nodes distributed over 286 sites. Other measurement scenarios may lead to much higher numbers of identified neighbor pairs.

Verification Using Average End-to-end Round-Trip Time A second way of verifying the results of cluster identification is to compare the average round-trip times between identified neighbors to the average round-trip times between other endpoint pairs. We have used the set of average round-trip times between each pair of endpoints as a reference. If the cluster identification method performs well the round-trip time between identified neighbors should be among the smallest in the reference set. We verify this using a percentile-percentile plot of the set of average round-trip times between identified neighbors and the reference set (Fig. 5.21).

We can clearly see that the round-trip times between identified neighbors are very small compared to the round-trip times between other pairs of endpoints. For $t_O = 0\%$, $b = 1\%$, and $t_B = 0.1\%$, 94% of the round-trip times between identified neighbors are smaller than the first percentile of the reference set. 100% are smaller than the second percentile. Even with $b = 5\%$, 88% of the identified neighbors have average round-trip times smaller than the second percentile of the reference set. For comparison, we have also included the plots for cluster identification without bias detection. The parameters were otherwise the same. Again we can see a positive impact of the bias detection function B on cluster identification. In the case $b = 5\%$, only 40% of the identified neighbors had average round-trip times smaller than the second percentile of the reference set, as compared to 88% with bias detection. Nevertheless, the results for $b = 1\%$ without bias detection are still rather good. 100% of the round-trip times between identified neighbors were smaller than the fifth percentile of the reference set.

Cluster Identification with Non-averaged Round-Trip Times

The results from Section 5.6.2 show that the presented method is able to identify clusters based on end-to-end round-trip time measurements performed from a single point of observation. However, the measurement values in the data

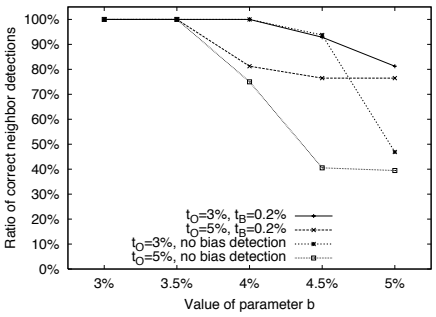


Figure 5.22: Ratio of confirmed neighbor detections for different parameter sets, non-averaged RTT values

represent the mean of ten measurements each, which significantly reduces their variance. Consequently, we have also evaluated the approach with non-averaged measurements.

We have gathered 25 two-day time series of round-trip time, measured using `ping`. The first halves of the time series were again used for cluster identification while the second halves were used for verification. We have selected 25 distinct endpoints from the sites of five universities as measurement peers. The round-trip times were measured every five seconds for 48 hours from a single endpoint at the University of Bern.

As in Section 5.6.2 we have applied the cluster identification procedure several times with different parameter sets. Verification was also done similarly. For each detected neighbor pair we have compared the second halves of the time series. If the values were within a 5% band of each other 95% of the time we would consider the detection confirmed. The results are shown in Fig. 5.22.

We observe that the ratio of confirmed neighbor detections also reaches 100% with non-averaged round-trip time values. However, the algorithm becomes more sensitive to changes in all three parameters. Fig. 5.22 shows that the ratio of confirmed neighbor detections rapidly decreases with rising values of b and t_O. On the other hand, the algorithm rejects all endpoint pairs with $b < 3\%$. Since outliers are much more frequent with non-averaged values, we also had to choose t_O greater than 0%. The increased variance of non-averaged measurement values thus effectively reduces the range of useful choices of parameters. The same effect can be observed for bias detection. Fig. 5.22 shows that cluster identification without bias detection results in significantly more errors. However, compared to the experiment with averaged round-trip values, the difference between both cases is much bigger. We conclude that the presented cluster identification approach is also useful for non-averaged measurements of round-trip time.

Cluster Identification with Available Bandwidth In Sections 5.6.2 and 5.6.2 we have evaluated the presented cluster identification method using measurements of round-trip time. Nevertheless, the method should also be able to detect clusters based on measurements of available bandwidth. We investigate this using a similar experiment as in Section 5.6.2. We have gathered sixteen 24-hour time series of available bandwidth between a single observation

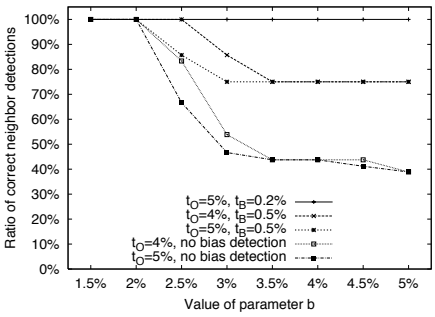


Figure 5.23: Ratio of confirmed neighbor detections with available bandwidth

point at the University of Bern and various endpoints in other universities' sites. The available bandwidth was estimated by downloading a sufficiently large file via HTTP, which was repeated every five minutes during 24 hours. We have used TCP throughput as an estimate for available bandwidth since it does not require any special software on the measurement peers. The results presented in [53] show that TCP throughput is a usable estimate of available bandwidth.

As in the previous experiments we have used the first halves of the time series for cluster identification and the second halves for verification of the results. However, we have used different parameter values than with round-trip time measurements for the verification because of the high variance of the measurements. We have reduced the threshold for outliers to 2% but have increased the band b to 10%. Note that cluster detection used transformed values as described in Section 5.4.1 while the verification was done using the original values.

Fig. 5.23 shows the results of the experiment. Again, the method is able to reach a 100% ratio of confirmed neighbor detections. However, the parameter t_B (bias threshold) has much more impact than with round-trip times. With $t_B = 0.2\%$ the ratio of confirmed detections was constantly 100%. Increasing it to $t_B = 0.5\%$ resulted in a 25% smaller ratio for $b > 3.5\%$. Without any bias detection the ratio of confirmed neighbor detections even dropped below 50% in some cases. Another difference to the previous experiments is the effect of parameter b, which decreases for values greater than 3%. With round-trip times it grew with higher values. The choice of parameter t_O has an influence similar to the one for non-averaged measurements of round-trip time.

5.7 Prototype Implementation and Tests

For the purpose of testing and evaluating the architecture described in this Chapter we have implemented a prototype. The implementation was mainly done using the Java 1.4.2 programming language and framework [55]. Furthermore, we have included the Java-based FreePastry implementation available from [34] for its DHT functionality, and the Java Matrix Package (JAMA) [48] for its linear equation solver used in the computation of prediction models. XMLRPC communication is based on the Apache implementation [141]. Group leaders also use the C++ implementation of Meridian available from [138].

5.7.1 Prototype Design

For most tasks the implementation relies on a single Java application whose design is a layered set of packages as shown in Figure 5.24. A package may only access the interfaces of other packages if they belong to a layer lower than its own. As long as they follow this rule packages may also call functions of packages more than one layer below their own. The packages on the interface layer provide the APIs to client processes, other group members, and for inter-group communication. The driver layer provides the intelligence behind the interface layer. It contains a "driver" for every role a node may have to fill. On the model layer we find the packages concerned with keeping track of the structure of the local group and of the network. Finally, functionality that is of general use can be found on the utility layer. We will discuss each layer in turn, starting at the top layer.

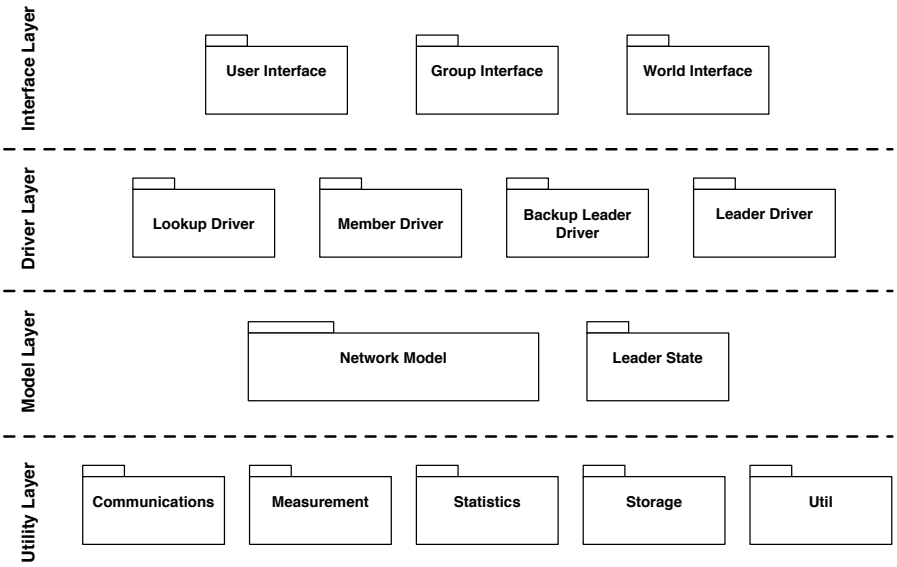


Figure 5.24: The four layers of the prototype design

Interface Layer

Every member node of the distance prediction service has three separate external interfaces, each implemented in a package.

The *user interface* provides functions through an XMLRPC interface for client processes running on the same host as the service. Using the `measure` function, a client process can schedule a measurement to a given target host. A measurement specification object describes the measurement target, type, and duration as well as the desired frequency of the measurement. The function blocks until the measurement has been made and then returns a vector of results. Errors are signaled by returning a single negative value. This is because we assume that distance measurements always yield positive values. If a client process has made measurements on its own, for example by monitoring its

own traffic, it can submit these measurements with a call to `addMeasurement`, specifying the target IP address, the time of observation, and the type of the measurements. The main functionality of the service is covered by `predict`. Similar to `measure`, this function expects an object describing the desired target, type, and time prediction interval, called a prediction specification. Both measurement specification and prediction specification objects are defined in the Network Model package.

Messages internal to a group are exchanged via the *group interface*. This includes functions for retrieving predictor objects to answer client requests, or storing new measurement data on the corresponding responsible node. These functions are called through Pastry's message passing mechanism, which routes a message object according to a cluster ID. When the message reaches the node responsible for the ID, it will be delivered to the appropriate function in the group interface. Further functions in this package are those concerned with group leadership. They accept keep-alive messages and leader state backups from the group leader. Finally, another set of functions receives backup data and messages for the migration or realignment of clusters.

XMLRPC messages from outside the group are handled in the *world interface* package. There are only two cases where this interface is used. New nodes can contact the group leader through this interface to check the grouping criterion and possibly join the group. Additionally, when a remote host needs to determine the group ID of the local group it can send a request to this interface.

It should be noted that the actual communication does not take place on the interface layer. Instead, the user, group, and world packages simply register their available interface functions with the communications package on the utility layer, which takes care of any further details of communication.

Driver Layer

The packages on the driver layer implement the different roles a node can fill. After joining a group, every node utilizes the lookup driver for processing prediction and measurement requests, and the member driver for managing its membership in the Pastry ring. Group leaders and backup leader also use the leader driver and backup leader driver packages, respectively.

As the name implies, the main task of the *lookup driver* is to answer lookup messages coming from the Pastry ring. As such it serves as a front end to the network model package on the model layer. However, it also maintains the integrity of the local node's part of the repository. From time to time it initiates a cluster realignment in the network model through the `maybeSplitCluster` function. Furthermore, it regularly creates cluster info objects from the clusters stored on the local node and broadcasts them to other group members. The lookup drivers on the receiving end compare these objects to their own clusters and initiate a merge operation if necessary. The cluster migration required during merging and splitting, as well as the transmission of cluster backups, is also part of the lookup driver's responsibilities.

The *member driver*'s function is simple. It returns the local group's ID when queried through the world interface, and it implements the group locating procedure executed when the node joins the distance prediction service network.Once

the node finds a group, the driver contacts the group leader to join the Pastry ring. After joining, the FreePastry implementation takes care of the ring management itself.

The *leader driver* implements the group leader's part of the joining procedure. When queried it returns a list of neighbors enabling the joining node to check the grouping criterion. If the node decides to join the group it provides it with the information necessary to connect to the Pastry ring. The leader driver package also controls the Meridian process running on group leader nodes. Because the leader of a group might fail it regularly sends replica of its current state to the backup leaders. Additionally, it broadcasts keep-alive messages to the group to signal that it did not fail yet. On the backup leaders, the *backup leader driver* takes care of storing the state replica and keeping track of the keep-alive messages. If a backup leader driver does not receive any keep-alive messages for a certain amount of time (which varies depending on the node's position in the backup leader chain) it promotes its node to leader, restores the latest leader state, and starts a Meridian process.

Model Layer

The model layer is where information about known hosts, groups, and clusters is handled. The state object used by the leader driver and backup leader driver is also defined here.

The *network model* package offers a large unified interface for all operations concerning hosts, groups, and clusters (in cases where we do not make a distinction between hosts, groups, and clusters we refer to them as *network entities*). Its functionality can be divided into four categories. The first category provides functions that can be called to find out the current state of the local node's part of the repository. Packages from upper layers can inquire about the known cluster, group, and host IDs, or directly ask whether a given network entity is stored at this node.

The second category is concerned with the performing and storing of measurements as well as the computation of prediction models and retrieval of distance predictors. We concentrate these functions in the network model interface because requests pertaining to measurements and predictions generally influence the state of the network model. The frequency of requests regarding a cluster indicate its popularity and therefore the likelihood of it being deleted when run short of storage space. Moreover, adding new measurements to a cluster may trigger a re-computation of its prediction model or the deletion of old data. Finally, when the package registers frequent prediction requests coinciding with a lack of new measurements it may decide to schedule automatic measurements to fill the gap.

The third category includes all functionality for creating, destroying, and migrating clusters. A new cluster can be created either by `createClusterFromHost` or `createClusterFromGroup`, depending on the type of the cluster's initial member. Removing a cluster can be done with the `removeNetworkEntity` function, which can also be used to remove hosts and groups from their respective clusters. For cluster migration and backups, the network model package can create *cluster packages* containing the current state of a cluster, its host and group IDs, and all measurement data andmodels re-

lated to it. The package includes enough information to restore the cluster data structure on another node. It is thus suitable for both, backups and cluster migration.

Functions for merging and splitting of clusters are compiled in the fourth category. When a clusters are merged or split, the network model package blocks all further calls for the duration of the operation and then computes a new clustering for the concerned hosts and groups. After this, it rearranges the clusters in the local repository and returns a list of all affected cluster IDs to the caller. The caller (i.e. the lookup driver) will then select the clusters with an ID in other nodes' areas of responsibility and migrate them to these nodes. Finally, it will update the entries in the Pastry ring to reflect the changes.

The network model package does not implement any of the algorithms needed for the above operations but rather relies on the tools from the measurement, statistics, and storage packages on the utility layer. However, since these packages cannot interact, the network model package coordinates the data and algorithms provided by each.

Utility Layer

Algorithms and tools used in higher layers of the architecture are located in various packages on the utility layer.

The *communications* package contains the objects for sending and receiving messages using either XMLRPC or Pastry. Each interface on the top layer corresponds to a communicator object in this package. Communicator objects do not provide any external API by themselves. Instead, the interface packages register their functions with the communications package using an identification string and a callback object. The strings are then used to advertise the interface functions to other processes and nodes. When a message arrives at the communications package it will be parsed and mapped to one of the interface functions and subsequently sent to the corresponding package through the registered callback object.

Measurement tools of arbitrary types are implemented in the *measurement* package. Each type of measurement is represented by a type of meter object. Depending on the their type, meter objects may either implement a measurement technique themselves, or they may delegate the actual measurement to an external process such as `ping`, in which case they only parse the output of the process. Meters are in turn controlled by measurement gatherer objects whose task it is to collect the meter's measurements and forward them to the network model package, where they will be delivered to the requesting client or stored in the repository. Measurement gatherers also detect erroneous meters and terminate them accordingly.

Algorithms for neighbor detection and clustering as well as the computation of prediction models are compiled in the *statistics* package. It also contains the time series class used as input to the algorithms. However, since this package has no access to the storage it relies on the network model package to provide the necessary data. The algorithms themselves are described in 5.4 and A.2.

The *storage* package manages the files on the node's harddisk and provides functions for creating, deleting, and changing them. It serializes measurement data and models using different file formats, wrapped by series file and model file objects, respectively. To improve the performance of file system access the

package implements a transparent caching mechanism based on a write-through strategy. When we read from a file the storage package also keeps a cached copy of the data. However, write operations are always carried out without caching. The storage package also aims to keep the number of open files to a minimum in order not to exceed the operating system's open file limit. Files always appear open to a client. However, the storage package detects when a file goes out of use and automatically closes it. When it receives a read or write request it transparently reopens the file.

The *util* package finally is a collection of helpful classes used in many places throughout the prototype implementation. Examples are exception classes for error notification, read/write locks for synchronizing file access, and timer mechanisms to defer the execution of a function to a later time.

5.7.2 Tests

In this section we present the results of a test deployment of the distance prediction service's prototype implementation. The tests were performed on PlanetLab [95].

Test Setup

We have deployed the distance prediction service on eleven PlanetLab nodes at the University of California at Berkeley. All eleven nodes formed a single peer-to-peer group that received prediction requests generated by a controlling node. Each node had a Pentium 4 CPU running with 3 GHz.

Since our distance prediction service relies heavily on user requests and observing user traffic, a fundamental problem for testing a system like this is to generate realistic user requests and traffic. We believe that the request pattern observed by a real deployment of our service would resemble the request pattern observed by a web proxy, because we expect that a large part of the requests pertain to a very small set of popular destination addresses. Nevertheless, the number of requests to a proxy should be higher since loading a web page normally involves several requests sent shortly after each other. In contrast, a client would request a distance prediction only once for a given network connection.

Accordingly, we have based our tests on a 24-hour log from the web proxy server of the University of Bern. Since the use of this proxy is mandatory, the log shows the web requests of all end systems within the University of Bern during 24 hours. The log records the time and destination of each web request on a single line in a text file. We have determined the distribution of destination IP addresses separately for each hour of the log and have used it to generate distance prediction requests. During each hour of the test, the controlling node generated distance prediction requests according to the distribution of the corresponding hour of the web proxy's log. Thus, popular destinations have a higher chance of being requested.

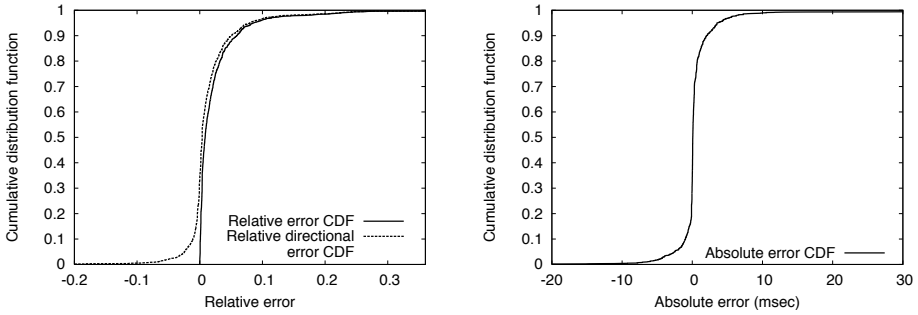
Unfortunately, the log entries do not contain any usable information about the observed round-trip time or throughput. Therefore, we were unable to emulate the monitoring of existing user traffic that would be done in a real world deployment. We have thus restricted the test to the prediction of round-trip times. In order to replace the missing passive measurements we have made active

`ping` measurements to the requested destination IP addresses. We scheduled such measurements once every 30 seconds for destinations that were lacking recent measurement data.

In the tests, each group member received a prediction request once every 20 seconds. Each successful distance prediction also triggered an additional round-trip time measurement to determine the error of prediction. Cluster information was broadcasted by each group member every 60 seconds to detect possibilites to merge clusters (the prototype did not include the rate adaption methods described in Section 5.5.3). Group members sent backup data to their neighbors every 5 minutes. We have run two tests. In the first test we have focused on the predictions and clustering made by the 11 nodes. In the second test we have monitored the network traffic and CPU loads of 3 nodes. Both ran for 4 hours, using the daytime hours 8–12 from the proxy log. Measurement data in the repository was stored for 2 hours before discarding it.

Test Results

The first test focused on the predictions and clustering made by the system. Naturally, one of the most important aspects of a distance prediction service is the error of prediction. Two widely used figures for this error are the *relative error* and the *directional relative error*, both of which are defined in [82]. The relative error of prediction is defined as $|\hat{x}-x|/\min(\hat{x},x)$ (where $\hat{x}$ is the predicted value), and the directional error of prediction is defined as $(\hat{x}-x)/\min(\hat{x},x)$. Furthermore, we look at the absolute error of prediction $\hat{x}-x$.



(a) Relative error and directional relative error

(b) Absolute error

Figure 5.25: CDFs of relative and absolute error of prediction

Figures 5.25(a) and 5.25(b) show the corresponding cumulative distribution functions. The observed results are very encouraging. Only very few predictions significantly over- or underestimate the measured round-trip time. Moreover, there is no noticeable bias. The results are significantly better than those achieved with GNP or IDMaps in [82]. Unfortunately, they are not fully comparable since the scenarios differ and our service only provides distance predictions if it has recent measurement data available. If there is insufficient data in the repository, the service is forced to actively measure the distance. Nevertheless, if data is available the predictions are very good.

Using only the data available in the repository, the group members were able to make predictions in 55.08% of the cases, which is a rather small ratio.

The main reason for this is the relatively low rate of requests used in the test. Analysis of the web proxy logs we used shows that the ratio of successful predictions increases with the rate of requests. Figure 5.26 shows the results of this analysis. Each graph in this Figure shows the theoretically achievable percentage of successful predictions for a certain rate of requests. The six graphs show the results for the cases where information about a destination is kept in the repository for time interval between 15 minutes and 24 hours. As the Figure shows we can already achieve a success ratio of ca. 80% with request rate of 2 requests per second. Success ratios of over 90% can be reached with a request rate of 10 requests per second. Considering our test setup where each node receives a request every 20 seconds, this corresponds to groups of 40 and 200 members, respectively.

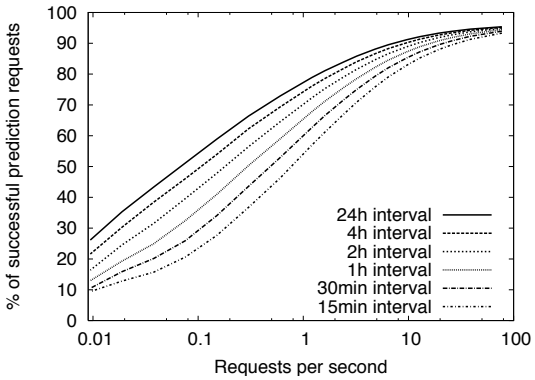


Figure 5.26: Ratio of successful predictions versus the rate of requests

Nevertheless, the ratio of successful predictions in our test is still significantly lower than the expected value of 66.34%. This is because many of the requested destinations did not reply to round-trip time measurements and had to be removed from the test. Since this problem mainly occurred with popular destinations the test had a more widespread distribution of destination addresses than the original web proxy log, resulting in a lower success ratio. Fortunately, this problem would not occur in a real deployment of the service since there, measurement data can be gathered from existing traffic and only rarely needs to be actively done.

A further result of interest is the number and the size of clusters the system has been able to identify during the test. Figure 5.27 shows the number of hosts assigned to clusters of various sizes. We can see that even though the majority of hosts are assigned to small clusters, clusters containing more than 80 hosts have been identified. In comparison to storing data for each destination independently, this clustering resulted in 62.06% fewer repository entries. This demonstrates that clustering may in fact significantly reduce the number of repository entries.

In order to be able to run the distance prediction service in the background on normal end systems such as workstations, the CPU, network, and storage space requirements of the service should be very moderate. The observed average storage space per node was 7.66 MBytes, which is an amount that should not pose any problems to modern computers. Since the service discards old

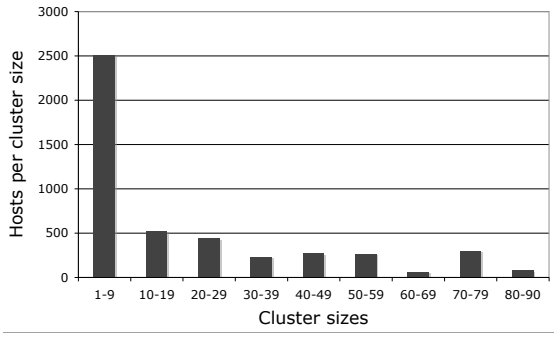


Figure 5.27: The number of nodes assigned to clusters of a given size

data after a certain time (3 hours in the test) we can assume that this number will remain approximately constant if the system runs longer. We have monitored the CPU usage and network traffic during a second test, by recording `top` (batch mode) and `tcpdump` output, respectively. The observed average CPU usage during this test was 4.00%, and the generated bandwidth per node was 30.18 kbps. Especially the second value is rather high. However, we believe that these values are mainly due to the short intervals at which clustering was updated and backup data was sent to other group members (1 and 5 minutes, respectively). By making these intervals larger and by implementing the adaptation rules for the cluster update rates described in Section 5.5.3 we should be able to significantly reduce the CPU and bandwidth requirements of the system without noticeably impairing the quality of the results.

Chapter 6

Conclusions

In this work we have studied two aspects of the prediction of Internet characteristics for distributed applications in the face of the Internet's considerable size and complexity. The first aspect concerns the prediction of the behavior of distributed applications that use a static deployment using scalable simulation techniques. Traditional tools for this purpose, such as sequential packet-based simulation, fail due to the complexity of Internet scenarios. We have investigated existing options for making simulation more scalable, and we have presented a novel hybrid simulator for inter-domain scenarios. The second aspect concerns the adaptivity of more dynamic applications, such as peer-to-peer networks, to the changing end-to-end characteristics of the network. We have studied numerous mechanisms and services helping applications detect and react to changing conditions. On the basis of our observations we have developed an end-to-end distance prediction service with several unique features.

The size of the Internet and the amount of traffic it carries poses a fundamental challenge to simulations of Internet scenarios. Researchers have approached this scalability problem either by making simulators more powerful through parallelization, or by using more abstract network models to make simulation more efficient but less accurate. In Chapter 2 we have presented existing work from both areas of research. Both, parallelization and abstraction, perform significantly better than conventional simulation methods when applied to large scenarios. However, the computational complexity is not the only problem we face when simulating Internet scenarios. Obtaining detailed information about Internet topology and traffic patterns tends to be challenging since the Internet consists of a vast number of independent administrative domains, which makes sufficiently detailed and accurate Internet scenarios hard to come by. In many cases we have to resort to coarse-grained scenarios inferred by measurement tools such as topology detectors and traffic meters. While this may be a severe problem for parallel simulation, we are often able to create meaningful simulation scenarios using abstract network models even if the available information is coarse-grained. Unfortunately, abstract models are normally tailored to specific aspects of the network or to special kinds of topologies, and the resulting simulations have a very limited scope.

In our hybrid simulator, described in Chapter 4, we combine detailed packet-based simulation with arbitrary network models through a plug-in mechanism. Thus, we can simulate different parts of the same scenario using different net-

work abstractions, taking advantage of the strong points of each. This is especially useful for combining detailed simulation of a distributed application with a coarse-grained model of the network. The hybrid simulator was implemented by extending the packet-based ns-2 network simulator. We have combined ns-2's packet-based models with an abstract network model for the scalable simulation of inter-domain networks. It is based on the observation that the internal networks of Internet service providers (domains) are often free of congestion while queuing delay and packet losses are caused by the inter-domain links between them. The model combines methods from analytical queuing theory and time stepped fluid simulation. The hybrid simulator has been integrated into a large architecture for the monitoring, modeling, simulation, and visualization of inter-domain networks. The architecture allows for constructing simulation scenarios based on measurements coming from the monitoring tools. Users can then add further changes before starting the simulator. The communication between the architecture and the simulator relies on XML messages that contain the necessary network topology and traffic traces. Auxiliary tools translate these messages into simulation scripts, run the simulator, convert the results to a standard format, and then return them.

In a second part of the work we have focused on predicting end-to-end Internet characteristics in order to help distributed applications like peer-to-peer networks adapt to the properties of the underlying physical network. We have identified shortcomings of the various existing approaches presented in Chapter 3, and we have described a novel distance prediction service in Chapter 5, which addresses these shortcomings. Existing approaches often rely on additional infrastructure in the network, or they require a rather large number of participants at distributed locations in the network in order to be able to provide useful distance estimates. Both properties are hurdles for the deployment of these approaches. Many approaches are also restricted to estimating only the average round-trip time between member nodes of the service, making them impractical for many application scenarios. Furthermore, most distance estimation services ignore the variability of Internet characteristics and the multitude of possible end-to-end distance measures.

In contrast to the majority of the existing distance estimation schemes, our approach considers series of distance measurements instead of single, averaged values. This allows for identifying trends and variances of the observed distances and therefore makes it possible to provide more sophisticated distance predictions. The necessary measurements are mainly obtained by monitoring application traffic. This is practical since we can assume that client applications request predictions for the same Internet hosts that they communicate with. Thus, rather than restricting the service to predict distances between member nodes only, we predict distances to the Internet hosts that the clients are actually interested in. Since users from the same network location tend to be interested in the same resources on the network, our distance prediction service was based on creating groups of nodes from the same network locations. Inside the groups the nodes exchange end-to-end measurements and predictions and distribute the workload of computing predictions through a local peer-to-peer network. Measurement data and prediction models are stored in a distributed local repository. We have given special focus to making the storage of measurement data in the repository efficient. Measurements to remote hosts are often very similar if the measurement targets are located close to each other. In such

cases, we can consider the remote hosts a single unit, called a cluster. Since the number of clusters is much smaller than the number of hosts this scheme can significantly reduce the workload on the repository. In Chapter 5 we have described an algorithm for cluster identification based on detecting similarities between measurement series to different remote hosts. Furthermore, we have developed an algorithm for the automatic organization of nodes into groups that does not require any prior knowledge of the nodes' network locality. It uses the nearest neighbor search provided by the existing Meridian framework. In another part of Chapter 5 we have described the architecture of the distance prediction service and the design of a prototype implementation. The algorithms for the identification of remote clusters and the creation of peer-to-peer groups have been evaluated using simulations based on measurement data from PlanetLab, a large testbed of computers distributed throughout the Internet. PlanetLab was also used to test the prototype implementation and evaluate its performance.

Our distance prediction service has several advantages. It does not require any additional infrastructure in the network due to its peer-to-peer design. The necessary software can be installed on conventional personal computers connected to the Internet. Because of its focus on groups of nodes from a single network location it can be deployed at a single site while still being able to provide distance predictions to hosts anywhere in the Internet. Moreover, the distance predictions it computes are more sophisticated than the simple estimates of existing approaches, and they can be applied to a multitude of different distance measures such as available bandwidth and delay jitter.

Appendix A

Mathematical Notes

A.1 Proof of the Impossibility to Embed an "Y" Topology in Euclidean Space

In Section 3.4.3 we have stated that it is impossible to embed a "Y" topology into Euclidean space without error, if all links in the topology have positive length. The proof is based on the Cayley-Menger determinant [127], which calculates the squared volume of a d-simplex based on the distances between its points. It is defined as

$$V^2 = \frac{(-1)^{d+1}}{2d(d!)^2} \cdot \begin{pmatrix} 0 & 1 & 1 & 1 & \\ 1 & 0 & d(p_1,p_2)^2 & d(p_1,p_3)^2 & \cdots \\ 1 & d(p_2,p_1)^2 & 0 & d(p_2,p_3)^2 & \\ 1 & d(p_3,p_1)^2 & d(p_3,p_2)^2 & 0 & \\ & & \vdots & & \ddots \end{pmatrix} \tag{A.1}$$

where $d(p_i, p_j)$ is the distance between points p_i and p_j. If the squared volume V^2 is negative for a given set of distances, they cannot be used to construct a simplex and thus they cannot be embedded without error.

For our case of the "Y" topology shown in Figure 3.7(a) we have to consider a 3-simplex (a tetrahedron). We let $a = \overline{AB} > 0$, $b = \overline{BC} > 0$, and $c = \overline{BD} > 0$. The distances over two hops add up, and we get $\overline{AC} = a + b$, $\overline{AD} = a + c$, and $\overline{CD} = b + c$. The determinant of interest for this case is thus

$$V^2 = \frac{(-1)^{3+1}}{2 \cdot 3 \cdot (3!)^2} \cdot \det \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & a^2 & (a+b)^2 & (a+c)^2 \\ 1 & a^2 & 0 & b^2 & c^2 \\ 1 & (a+b)^2 & b^2 & 0 & (b+c)^2 \\ 1 & (a+c)^2 & c^2 & (b+c)^2 & 0 \end{pmatrix} \tag{A.2}$$

We find that $\det(\cdot) = -32 \cdot a^2 \cdot b^2 \cdot c^2$, which is always negative. Furthermore, $(-1)^{3+1}/2 \cdot 3 \cdot (3!)^2$ is positive. It follows that V^2 is negative, and thus that the distances of the "Y" topology cannot be embedded if $a, b, c > 0$.

A.2 AR Models and Yule-Walker Equations

The Yule-Walker equations are a method for estimating the parameters $\phi_{1\ldots p}$ of an auto regressive (AR) model using a linear system of equations. An AR model of order p has the following form

$$x_{t+1} = \phi_1 x_t + \ldots + \phi_p x_{t-p+1} + \varepsilon_t$$

where x_i are the values of the modeled time series, and ε_t is a random "shock" at time t. ε_t follows a normal distribution with mean 0.

To find the Yule-Walker equations we multiply the above equation with x_{t-k+1} (where k is the *lag*) and get

$$x_{t-k+1}x_{t+1} = \phi_1 x_{t-k+1}x_t + \ldots + \phi_p x_{t-k+1}x_{t-p+1} + x_{t-k+1}\varepsilon_t$$

By taking expectancies we arrive at

$$\gamma_k = \phi_1\gamma_{1-k} + \ldots + \phi_p\gamma_{p-k}$$

where γ_i is the auto-covariance function of the time series with lag i. Note that since $E(\varepsilon_t) = 0$ the last term is also 0 and can be eliminated. We repeat this for $k = 1, \ldots, p$ to get the system of linear equations (i.e. the Yule-Walker equations)

$$\begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_p \end{pmatrix} = \begin{pmatrix} \gamma_0 & \cdots & \gamma_{p-1} \\ \vdots & \ddots & \vdots \\ \gamma_{p-1} & \cdots & \gamma_0 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_p \end{pmatrix} \tag{A.3}$$

This system is easily solvable using standard methods.

Bibliography

[1] Jong-Suk Ahn and Peter B. Danzig. Speedup vs. simulation granularity. *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.

[2] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *18th ACM Symposium on Operating System Principles (SOSP), Banff, Canada*, October 2001.

[3] Sigrún Andradóttir and Teunis J. Ott. Time-segmentation parallel simulation of networks of queues with loss or communication blocking. *ACM Transactions on Modeling and Computer Simulation*, 5(4):269–305, October 1995.

[4] D. Anick, D. Mitra, and M. M. Sondhi. Stochastic theory of a data-handling system with multiple sources. *The Bell System Technical Journal*, 61(8):1871–1894, October 1982.

[5] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu an Chen, Xiang Zeng, Jay Martin, and Ha Yoon Song. Parsec: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77–85, October 1998. ISSN 0018-9162.

[6] Rajive L. Bagrodia. Iterative design of efficient simulations using maisie. In Barry L. Nelson, W. David Kelton, and Gordon M. Clark, editors, *Proceedings of the 1991 Winter Simulation Conference*, pages 243–247, 1991.

[7] Paul Barford and Mark Crovella. An architecture for a www workload generator. In *Proceedings of the ACM SIGMETRICS*, 1998.

[8] Maurizio Bartoli, Florian Baumgartner, Christof Brandauer, Torsten Braun, Sandor Kardos, Fabrizio Orlandi, Matthias Scheidegger, and Jörn Seger. The intermon simulation framework. In *International Workshop on Inter-Domain Performance and Simulations (IPS2004), Budapest, Hungary*, pages 130–138, March 2004.

[9] Florian Baumgartner, Matthias Scheidegger, and Torsten Braun. Enhancing discrete event network simulators with analytical network cloud models. In *International Workshop on Inter-domain Performance and Simulation (IPS), Salzburg, Austria*, pages 21–30, February 2003.

[10] Florian Baumgartner, Matthias Scheidegger, and Torsten Braun. Simulating router- and domain characteristics. In *International Workshop on*

Inter-Domain Performance and Simulations (IPS2004), Budapest, Hungary, pages 139–145, March 2004.

[11] Stephan Bohacek, João P. Hespanha, Junsoo Lee, and Katia Obraczka. A hybrid systems modeling framework for fast and accurate simulation of data communication networks. In *Proceedings of the ACM SIGMETRICS'03, San Diego, USA*, pages 58–69, June 2003.

[12] 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. *IEEE Computer*, 33(5):59–67, May 2000.

[13] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting, 2nd Edition*. Springer, March 2003. ISBN 0-38795-351-5.

[14] Randy Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[15] Miguel Castro, Peter Druschel, Anne-Marie Mermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-bandwidth multicast in cooperative environments. In *ACM Symposium on Operating System Principles (SOSP), New York, USA*, October 2003.

[16] K. M. Chandy and R. Sherman. Space-time and simulation. In *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, pages 53–57, March 1989.

[17] Xinjie Chang. Network simulations with OPNET. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W Evans, editors, *Proceedings of the 1999 Winter Simulation Conference*, 1999.

[18] Ian Clarke, Theodore W. Hong, Scott G. Miller, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[19] M. Coates, R. Castro, R. Nowak, and Y. Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. *ACM SIGMETRICS Performance Evaluation Review*, 30(1):11–20, June 2002.

[20] M. J. Coates, A. Hero, R. Nowak, and B. Yu. Internet tomography. *IEEE Signal Processing Magazine*, May 2002.

[21] J. W. Cooley and J. W. Tukey. An algorithm for machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965.

[22] Manuel Costa, Miguel Castro, Antony Rowstron, and Peter Key. PIC: Practical internet coordinates for distance estimation. In *24th International Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan*, 2004.

[23] James H. Cowie, David M. Nicol, and Andy T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50, 1999.

[24] Mark E. Crovela and Azer Bestavros. Self-similarity in world wide web traffic; evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[25] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW client-based traces. Technical report, Computer Science Department, Boston University, July 1995. BU-CS-95-010.

[26] Frank Dabek, Russ Cox, Frans Kasshoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM'04, Portland, USA*, August 2004.

[27] Peter B. Danzig, Sugih Jamin, Ramón Cáceres, Danny J. Mitzel, and Deborah Estrin. An empirical workload model for driving wide-area TCP/IP network simulations. *Internetworking: Research and Experience*, 3:1–26, 1992.

[28] N. G. Duffield and F. L. Presti. Network tomography from measured end-to-end delay covariance. *IEEE/ACM Transactions on Networking*, 12(6):978–992, December 2004.

[29] Daniel R. Figueiredo, Benyuan Liu, Yang Guo, Jim Kurose, and Don Towsley. On the efficiency of fluid simulation of networks. *Computer Networks*, 50(12):1974–1994, August 2006. ISSN 1389-1286.

[30] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1993.

[31] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001. ISSN 1063-6692.

[32] P. Francis. A call for an internet-wide host proximity service (HOPS). White paper, March 1997.

[33] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: A global internet host distance estimation service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, October 2001.

[34] Freepastry. http://freepastry.com.

[35] R. Fujimoto, C. Cooper, and I. Nikolaidis. Parallel simulation of statistical multiplexers. In *32nd IEEE Conference on Decision and Control*, 1994.

[36] Luis Garcés-Erice, Ernst W. Biersack, and Pascal A. Felber. MULTI+: Building topology-aware overlay multicast trees. In *Fifth International Workshop on Quality of Future Internet Services (QofIS)*, September 2004.

[37] Michael R. Garrey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, USA, 1979. ISBN 0-71671-044-7.

[38] T. J. Giuli and Mary Baker. Narses: A scalable flow-based network simulator. Technical Report arXiv:cs.PF/0211024 v1, Standford University, USA, November 2002.

[39] The annotated Gnutella protocol specification. http://rfc-gnutella.sourceforge.net/developer/stable/index.html.

[40] A. G. Greenberg, I. Mitrani, and B. Lubachevsky. Unbounded parallel simulations via recurrence relations. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM, New York, 1990.

[41] Albert G. Greenberg, Boris D. Lubachevsky, and David M. Nicol. Efficient massively parallel simulation of dynamic channel assignment schemes for wireless cellular communications. In *Proceedings of the Eighth Workshop on Parallel and Distributed Simulation, Edinburgh, Scotland, United Kingdom*, pages 187–197, 1994. ISSN 0163-6103.

[42] Yang Guo, Weibo Gong, and Don Towsley. Time-stepped hybrid simulation (tshs) for large scale networks. In *Proceedings of the IEEE Infocom*, 2000.

[43] Pedro A. Aranda Gutiérrez and Ilka Miloucheva. Integrating inter-domain routing analysis in novel management strategies for large scale ip networks. In *Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04), St.Petersburg, Russia*, February 2004.

[44] Fang Hao and Pramod Koppol. An internet scale simulation setup for BGP. *ACM SIGCOMM Computer Communications Review*, 33(3), July 2003.

[45] Fang Hao, Ioanis Nikolaidis, and Ellen W. Zegura. Efficient simulation of ATM networks with accurate end-to-end delay statistics. In *International Conference on Communications*, pages 1799–1804, 1998.

[46] Fang Hao, Karen Wilson, Richard Fujimoto, and Ellen W. Zegura. Logical process size in parallel simulations. In *Winter Simulation Conference*, pages 645–652, 1996.

[47] P. Heidelberger and H. S. Stone. Parallel trace-driven cache simulation by time partitioning. Technical Report RC-15500, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, 1990.

[48] Joe Hicklin, Cleve Moler, Peter Webb, Ronald F. Boisvert, Bruce Miller, Roldan Pozo, and Karin Remington. JAMA: A Java matrix package. http://math.nist.gov/javanumerics/jama.

[49] S. M. Hotz. *Routing Information Organization to Suport Scalable Interdomain Routing with Heterogeneous Path Requirements*. PhD thesis, University of Southern California, 1994. Draft.

[50] Yang hua Chu, S. G. Rao, S. Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, October 2002. ISSN 0733-8716.

[51] Polly Huang, Deborah Estrin, and John Heidemann. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, pages 241–248, 1998.

[52] IST Intermon homepage. http://www.ist-intermon.org.

[53] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Transactions on Networking*, 11(4):537–549, August 2003.

[54] John Jannotti, David K. Gifford, and Kirk L. Johnson. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000)*, October 2000.

[55] Sun developer network. http://java.sun.com.

[56] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[57] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

[58] Kevin G. Jones and Samir R. Das. Time-parallel algorithms for simulation of multiple access protocols. In *9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2001.

[59] B. Kantor and P. Lapsley. RFC 977: Network news transfer protocol: A proposed standard for the stream-based transmission of news, February 1986. Status: PROPOSED STANDARD.

[60] G. Kesidis, A. Singh, D. Cheung, and W. W. Kwok. Feasibility of fluid event-driven simulation for ATM networks. In *Proceedings of the IEEE Globecom*, November 1996.

[61] Hwangnam Kim. Integrating network-calculus-based simulation with packet-level simulation for TCP operated networks. *Computer Networks*, 50(12):1995–2012, August 2006.

[62] Alexander Klemm, Christoph Lindemann, and Marco Lohmann. Modeling IP traffic using the batch Markovian arrival process. *Performance Evaluation*, 52:149–173, 2003.

[63] J. Klensin, N. Freed, M. Rose, E. Stefferud, and D. Crocker. RFC 1651: SMTP service extensions, July 1994. Obsoleted by RFC1869, STD0010 [64, 100]. Obsoletes RFC1425 [65]. Status: DRAFT STANDARD.

[64] J. Klensin, N. Freed, M. Rose, E. Stefferud, and D. Crocker. RFC 1869: SMTP service extensions, November 1995. See also STD0010 [100]. Obsoletes RFC1651 [63]. Status: STANDARD.

[65] J. Klensin, WG Chair, N. Freed, M. Rose, E. Stefferud, and D. Crocker. RFC 1425: SMTP service extensions, February 1993. Obsoleted by RFC1651 [63]. Status: PROPOSED STANDARD.

[66] Andreas Kock, Matthias Scheidegger, and et al. Integration report. Intermon Deliverable 16, October 2003.

[67] Krishnan Kumaran and Debasis Mitra. Performance and fluid simulations of a novel shared buffer management system. *ACM Transactions on Modeling and Computer Simulation*, 11(1):43–75, January 2001.

[68] Jonathan Ledlie, Peter Pietzuch, and Margo Seltzer. Stable and accurate network coordinates. In *26th International Conference on Distributed Computing Systems, Lisboa, Portugal*, July 2006.

[69] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic. In *ACM SIGCOMM, San Francisco*, pages 183–193, 1993.

[70] Hyuk Lim, Jennifer C. Hou, and Chong-Ho Choi. Constructing internet coordinate system based on delay measurement. In *Internet Measurement Conference 2003, Miami, USA*, October 2003.

[71] Yi-Bing Lin and Edward D. Lazowska. A time-driven algorithm for parallel simulation. *ACM Transactions on Modeling and Computer Simulations*, 1(1):73–83, 1991.

[72] Benyuan Liu, Daniel R. Figueiredo, Yang Guo, Jim Kurose, and Don Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Proceedings of the INFOCOM 2001, Anchorage, USA*, volume 3, pages 1244–1253. IEEE, 2001. ISBN 0-7803-7016-3.

[73] Benyuan Liu, Yang Guo, Jim Kurose, Don Towsley, and Weibo Gong. Fluid simulation of large scale networks: Issues and tradeoffs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99*, 1999.

[74] Bruce A. Mah. An empirical model of HTTP network traffic. In *Infocom'97*, pages 592–600, 1997. ISBN 0-8186-7780-5.

[75] Mohammad Malli, Chadi Barakat, and Walid Dabbous. CHESS: An application-aware space for enhanced scalable services in overlay networks. under submission.

[76] Friedemann Mattern. Efficient algorithms for distributed snapshots and global virtual time approximations. *Journal of Parallel and Distributed Computing*, 18(4), 1993.

[77] Petar Maymounkov and David Mazières . IPTPS 2002: 53-6. Kademlia: A peer-to-peer information system based on the XOR metric. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, 2002.

[78] Duncan C. Miller and Jack A. Thorpe. SIMNET: The advent of simulator networking. *Proceedings of the IEEE*, 83(8), August 1995.

[79] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1), March 1986.

[80] K. Moore. SONAR – a network proximity service. Internet-Draft, August 1998.

[81] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[82] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *IEEE Infocom*, 2002.

[83] T. S. Eugene Ng and Hui Zhang. A network positioning system for the internet. In *USENIX, Boston, USA*, June 2004.

[84] David Nicol and Philip Heidelberger. Parallel execution for serial simulators. *ACM Transactions on Modeling and Computer Simulation*, 6(3):210–242, July 1996.

[85] J. Oikarinen and D. Reed. RFC 1459: Internet Relay Chat Protocol, May 1993. Status: EXPERIMENTAL.

[86] Rong Pan, Balaji Prabhakar, Konstantinos Psounis, and Damon Wischik. SHRiNK: A method for scaleable performance prediction and efficient network simulation. In *IEEE Infocom*, 2003.

[87] Craig Partridge. RFC 974: Mail routing and the domain system, January 1986. See also STD0014 [88]. Status: STANDARD.

[88] Craig Partridge. STD 14: Mail routing and the Domain system, January 1986. See also RFC0974 [87].

[89] Vern Paxson. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1994.

[90] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[91] Kalyan S Perumalla, Richard Fujimoto, and Andrew Ogielski. TeD – a language for modeling telecommunication networks. *SIGMETRICS Performance Evaluation Review*, 25(4):4–11, 1998.

[92] Kalyan S. Perumalla, George F. Riley, Alfred Park, and Richard M. Fujimoto. Scalable RTI-based parallel simulation of networks. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS'03)*. IEEE, 2003.

[93] C. D. Pham, H. Brunst, and S. Fdida. Conservative simulation of load-balanced routing in a large ATM network model. In *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation (PADS 98), Banff, Canada*, pages 142–149, May 1998.

[94] Marcelo Pias, Jon Crowcroft, Steve Wilbur, Tim Harris, and Saleem Bhatti. Lighthouses for scalable distributed location. In *Second International Workshop on Peer-to-Peer Systems (IPTPS 2003), Berkeley, USA*, February 2003.

[95] Planetlab home page. http://www.planet-lab.org.

[96] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ninth annual ACM Symposium on Parallel Algorithms and Architectures, Newport, Rhode Island, USA*, pages 311–320, 1997. ISBN 0-89791-890-8.

[97] Anna L. Poplawski and David M. Nicol. Nops: A conservative parallel simulation engine for TeD. In *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation (PADS 98), Banff, Canada*, pages 180–187, May 1998. ISBN 0-8186-8457-7.

[98] J. Postel. RFC 788: Simple mail transfer protocol, November 1981. Obsoleted by RFC0821 [99]. Obsoletes RFC0780 [126]. Status: UNKNOWN. Not online.

[99] J. Postel. RFC 821: Simple mail transfer protocol, August 1982. See also STD0010 [100]. Obsoletes RFC0788 [98]. Status: STANDARD.

[100] Jonathan B. Postel. STD 10: Simple mail transfer protocol, August 1982. See also RFC0821, RFC1869. RFC974 [99, 64, 87]. Obsoleted by RFC2821. Obsoletes RFC788, RFC780, RFC772 [98, 126, 125].

[101] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C, 2nd Edition*. Cambridge University Press, 1992. ISBN 0-52143-108-5.

[102] random.org – True random number service. http://www.random.org.

[103] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM'01*, pages 161–172, August 2001.

[104] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Infocom 02*, 2002.

[105] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4), March 1995. Obsoletes RFC1654 [106]. Status: DRAFT STANDARD.

[106] Y. Rekhter and T. Lis. RFC 1654: A Border Gateway Protocol 4 (BGP-4), July 1994. Obsoleted by RFC1771 [105]. Status: PROPOSED STANDARD.

[107] George F. Riley. The georgia tech network simulator. In *Proceedings of the ACM SIGCOMM 2003 Workshops*, pages 5–12, 2003.

[108] George F. Riley, Richard M. Fujimoto, and Mostafa H. Ammar. A generic framework for parallelization of network simulations. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, College Park, MD, USA*, pages 128–135, 1999. ISBN 0-7695-0381-0.

[109] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM Internation Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany*, November 2001.

[110] Dan Rubenstein, James F. Kurose, and Donald F. Towsley. Optimistic parallel simulation of reliable multicast protocols. *SIGMETRICS Performance Evaluation Review*, 25(4):22–29, 1998.

[111] Kevin Savetz, Neil Randall, and Yves Lepage. *MBONE: Multicasting Tomorrow's Internet*. John Wiley & Sons, March 1996. ISBN 1-56884-723-8.

[112] Matthias Scheidegger, Florian Baumgartner, and Torsten Braun. Simulating large-scale networks with analytical models. In *Analytical and Stochastic Modelling Techniques and Applications (ASMTA2004), 18th European Simulation Multiconference (ESM 2004), Magdeburg, Germany*, pages 13–16, June 2004.

[113] Matthias Scheidegger, Florian Baumgartner, and Torsten Braun. An integrated simulator for inter-domain scenarios. In *Kommunikation in Verteilten Systemen 2005 (KiVS), Kaiserslautern, Germany*, pages 295–306, February 2005.

[114] Matthias Scheidegger, Florian Baumgartner, and Torsten Braun. Simulating large-scale networks with analytical models. *International Journal of Simulation Systems, Science & Technology Special Issue on: Advances In Analytical And Stochastic Modelling*, 6(1-2), January 2005. Invited journal version of the ASMTA paper.

[115] Matthias Scheidegger, Florian Baumgartner, and et al. Specification of the modelling and simulation toolkit. InterMON Deliverable 6, December 2002.

[116] Matthias Scheidegger, Florian Baumgartner, and et al. Integration of the inter-domain modelling and simulation toolkit. InterMON Deliverable 11, June 2003.

[117] Matthias Scheidegger and Torsten Braun. Improved locality-aware grouping in overlay networks. In *15. ITG/GI-Fachtagung Kommunikation in Verteilten Systemen (KiVS), Bern, Switzerland*, 2007.

[118] Matthias Scheidegger, Torsten Braun, and Florian Baumgartner. Endpoint cluster identification for end-to-end distance estimation. In *International Conference on Communications, Istanbul, Turkey*. IEEE, June 2006. CD-ROM.

[119] Matthias Scheidegger and et al. Evaluation of inter-domain QoS modeling, simulation and optimization. InterMON Deliverable 19, April 2004.

[120] Carsten Schmoll, Elisa Boschi, Florian Baumgartner, Matthias Scheidegger, and et al. Final architecture specification. Intermon Deliverable 15, April 2004.

[121] Secure hash standard. U.S. Department Commerce/NIST, National Technical Information Service, Springfield, VA, April 1995. FIPS 180-1.

[122] Puneet Sharma, Zhichen Xu, Sujata Banerjee, and Sung-Ju Lee. Estimating network proximity and latency. *ACM SIGCOMM Computer Communication Review*, 36(3):41–50, July 2006.

[123] Yuval Shavitt, Xiaodong Sun, Avishai Wool, and Bülent Yener. Computing the unmeasured: An algebraic approach to internet mapping. *IEEE Journal on Selected Areas in Communications*, 22(1):67–78, January 2004.

[124] Yuval Shavitt and Tomer Tankel. Big-bang simulation for embedding network distances in Euclidean space. In *IEEE Infocom 2003*, 2003.

[125] S. Sluizer and J. Postel. RFC 772: Mail transfer protocol, September 1980. Obsoleted by RFC0780 [126]. Status: UNKNOWN. Not online.

[126] S. Sluizer and J. Postel. RFC 780: Mail transfer protocol, May 1981. Obsoleted by RFC0788 [98]. Obsoletes RFC0772 [125]. Status: UNKNOWN. Not online.

[127] D. M. Y. Sommerville. *An Introduction to the Geometry of n Dimensions*. Dover Publications, New York, 1958.

[128] Sridhar Srinivasan and Ellen Zegura. M-coop: A scalable infrastructure for network measurement. In *Third IEEE Workshop on Internet Applications (WIAPP'03), San Jose, USA*, June 2003.

[129] Ion Stoica, Rober Morris, David Liben-Nowell, David R. Karger, M. Frans Kasshoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[130] Jeremy Stribling. All-pairs-pings for Planetlab. http://pdos.csail.mit.edu/~strib/pl_app.

[131] Boleslaw K. Szymanski, Adnan Saifee, Anand Sastry, Yu Liu, and Kiran Madnani. Genesis: A system for large-scale parallel network simulation. In *16th Workshop on Parallel and Distributed Simulations, Washington D.C., USA*. IEEE, May 2002.

[132] Wolfgang Theilmann and Kurt Rothermel. Dynamic distance maps of the internet. In *Infocom 2000*, 2000.

[133] Duc A. Tran, Kien A. Hua, and Tai Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Infocom 2003*, 2003.

[134] Brian W. Unger and Greg A. Lomow. A simulation environment for telecommunications. In G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, editors, *Proceedings of the 1993 Winter Simulation Conference*, pages 1152–1160, 1993.

[135] Anrdás Varga. OMNeT++ – portable simulation environment in C++. In *Annual Students' Scientific Conference (TDK), Budapest, Hungary*, 1992. In Hungarian.

[136] George Varghese and Tony Lauck. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. *ACM SIGOPS Operating Systems Review*, 21(5):25–38, November 1987. ISSN 0163-5980.

[137] Manuel Villén-Altamirano and José Villén-Altamirano. RESTART: A straightforward method for fast simulation of rare events. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 282–289, 1994.

[138] Bernard Wong. Meridian c++ code. http://www.cs.cornell.edu/People/egs/meridian/code.php.

[139] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: A lightweight network location service without virtual coordinates. *ACM SIGCOMM Computer Communication Review*, 35(4):85–96, October 2005. ISN 0146-4833.

[140] Hao Wu, Richard M. Fujimoto, and Mostafa Ammar. Time-parallel trace-driven simulation of CSMA/CD. In *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation (PADS'03)*, 1993.

[141] Apache XML-RPC. http://ws.apache.org/xmlrpc/.

[142] Anlu Yan and Wei bo Gong. Time-driven fluid simulation for high-speed networks. *IEEE Transactions on Information Theory*, 45(5):1588–1599, July 1999.

[143] G. U. Yule. On a method of investigating periodicities in disturbed series with special reference to Wolfer's sunspot numbers. *Philosophical Transactions Royal Society London Ser. A*, 226:267–298, 1927.

[144] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation (PADS 98), Banff, Canada*, pages 154–161, May 1998. ISBN 0-8186-8457-7.

[145] Xin Yan Zhang, Qian Zhang, Zhensheng Zhang, Gang Song, and Wenwu Zhu. A construction of locality-aware overlay network: mOverlay and its performance. *IEEE Journal on Selected Areas in Communications*, 22(1):18–28, January 2004.

[146] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

Acknowledgments

I would like to thank Fritz Bütikofer from the "Informatikdienste" of the University of Bern for providing the web proxy logs used in the evaluation of our distance prediction service implementation. Also, thanks go to Bernard Wong from Cornell University at Ithaca, USA, for advice about Meridian.

Curriculum Vitae

1974	Born on July 2, in Biel/Bienne, Switzerland
1981 – 1985	Elementary School Ipsach
1985 – 1990	Secondary School Nidau
1990 – 1995	Gymnasium Alpenstrasse Biel/Bienne, Typus E
1995 – 2001	University of Bern, Switzerland. Major in Computer Science and Minors in Mathematics and Political Economics
2001	M.Sc. in Computer Science, University of Bern
2001 – 2007	Ph.D. Student and Research Assistant at the Institute for Computer Science and Applied Mathematics, University of Bern