Infrastructure Aware Applications

Ph.D. Dissertation

by

Vilmos Bilicki

Supervisor

Dr. Márk Jelasity

Submitted to the

Ph.D. School in Computer Science

Department of Software Engineering

Faculty of Science and Informatics

University of Szeged

Szeged 2010

Contents

II The Applications 39

Acknowledgements

The writing of this thesis took me a lot of time and effort. This was not only my time but the time of my family. First, I would like to thank to my wife Andrea and my children Vilmos, Máté and András for their patience and support. I would also like to thank my father, mother and brother for all the support I have received from them. My father-in-law and mother-in-law helped our familiy a lot. Without the long brainstorming meetings we conducted with my supervisor Márk Jelasity and his keen insight into things, this work could not have been done. I also received encouragement from the leader of our department, Tibor Gyimóthy, who gave me useful suggestions and enough free time to be able to concentrate on this study. Next, I would like to express my gratitude to Márta Fidrich, with whom I discussed many aspects of my research. The former and current members of the Wireless Laboratory who also helped me a lot include: György Roszik, Péter Bagrij, Gábor Sey, Zoltán Sógor, József Dombi Dániel, Miklós Kasza, Vilmos Szűcs, Róbert Béládi, Ádám Végh and Gábor Molnár. I would like to thank them too.

My thanks also go to David P. Curley who zealously reviewed this work from a lingustic point of view.

I believe that things in life do not happen by chance, but are the result of the will of a higher entity. I am happy that He helped me set and achieve my goals.

1 Introduction

The success of the IP protocol stack and also the success of the Internet as a technology lies in its simplicity. Based on the classical hourglass visualization, a wide variety of first and second layer technologies are all related to the IP and the intelligence is provided by the higher layers in most cases on the edge of the network. In theory, the details of the infrastructure are hidden from the applications and the application programmers. The developer of a given application should focus on the implementation of their own business logic without being concerned about the lower layers. The network devices in the core of the network are simple and stateless, their main task being fast packet forwarding, while logically near the customers the active devices may provide some basic services for the network operator. This paradigm is about 30 years old and since its introduction many things have changed. Almost all the services related to information exchange are now based or will be based on the services provided by the IP network. The widespread use of overlay technologies and virtualization is also an important trend. This heterogeneous demand together with the wide variety of access layer technologies are the main facilitators of the paradigm shift we are now witnessing. The network is becoming evermore intelligent and providing context-based services, while the applications or the underlying layers are becoming increasingly infrastructure aware. The main active device vendors are now opening the black boxes and they starting to provide environments for running third party applications on the active devices [14]. This step will make the intelligent or active network described in [128] possible. Here the goal of the thesis is twofold. First, we would like to show that even the known intelligent services have serious scalability issues. We will show that a new aspect of the

number of unicast or multicast flows going through an active device should be considered by network engineers or application developers. Second, we will present four approaches where the application itself takes into the account the capabilities of the underlying network.

The novel features presented by this work are:

- The scalability issues of the intelligent services
- The ability to observe the hiding botnets on the Internet
- A small degree, robust DHT
- Signaling compression for the adaptation of the SIP protocol
- Distributed storage

Part I

The network

2
Network infrastructure

In this chapter we shall provide necessary background details to help the reader understand the motivation behind the infrastructure aware application presented in the second part. First, we will discuss the most important trends we should consider. Then a high level overview of the current network architectures will be given. Afterwards we will discuss the architectures and capabilities of the active devices operating in different layers, give a high level overview of the intelligent services available in the current network deployments and then study the scalability of these services via extensive measurements of real active devices. We will show that the number of flows should also be considered during the traffic engineering process or it should be considered by the application developers. As a significant percentage of the total IP traffic is produced by the P2P ecosystems, we will provide an overview of the current methods for network aware P2P applications (otherwise called ISP friendly P2P applications) found in the literature today and show that the current focus on the traffic volume should be supplemented with an additional metric; namely the number of flows reproduced by a node.

2.1 Trends

The IP network and the services offered on the top of the network are constantly evolving. Here we would like to give an overview of the most important trends we should consider when we discuss infrastructure aware applications.

2.1.1 Broadband end-users

Broadband end-users are the main facilitators of the evolution of the Internet. Based on a study [15], the consumers now produce 15 PByte per year, which is three times larger than the traffic generated by the business entities. In the future this difference is expected to increase. These users are a potential market for new services. The definition of broadband is also expected to change with the advent of fibre-to-home or fibre-to-basement installations. One important and special class of broadband end-users are the mobile end-users.

2.1.2 3G

Wired line networks have been used over for two decades now. These networks at the start had teething troubles, but now they are in a productive and well-tested state. We know a lot about wired network design and we have a large set of well tested and relatively cheap wired network elements (switches, routers), but the main thing is that these techniques are widely used. So it is not surprising that the 3rd Generation Partnership Project (3GPP) community chose the IP protocol as the backbone for future Universal Mobile Telecommunication System (UMTS). UMTS will be an all-IP solution. This is why the tendency in mobile core system developers is to change from their own individual protocols to the widely used Internet protocols. In 3G telephony the pace is increasing and new protocols are appearing. It is worthwhile doing this because the manufacturers have adopted common standards to make devices compatible, and many universities and companies can participate in the development of these protocols.

Based on a study [16], the amount of data generated by mobile end users is expected to double every year. By 2014 about 70% of the mobile users will probably use laptops or other mobile ready portables, while about 21% will probably use smart phones. These trends tell us that the issues with this access layer technology should be taken seriously by the application developers.

2.1.3 Services and applications

Based on a study [15], the amount of data produced by the video-on-demand service is expected to double every two years until 2014 and it will be responsible for about 60% of the total consumer traffic. P2P traffic will provide about 20% of the total traffic. The Video TV service will account for some 7% of the global consumer traffic. Here a large number of users will be connected to the same streaming channel and, in this case, the unicast data transfer model will not be an effective and scalable solution. Currently there are only a few applications which utilize the multicast support of the network, but the IPTV service could be one facilitator of the intra AS multicast infrastructure

deployment. On the Internet scale, P2P streaming solutions might become an alternative to the current cloud infrastructure. P2P video streaming currently accounts for 7% of the total P2P traffic.

2.1.4 IPv6 and multicast

It is well known that the IPv4 address space is a valuable resource. This is true for the Class D addresses as well. So it could happen that the Triple Play solutions will become the driving force behind IPv6. One of the most attractive features of the IPv6-based networks is their multicasting capability. Due to their large address space, many addressing solutions can be applied. The use of scoped addresses is another potential area for efficient traffic engineering. With efficient bandwidth usage we also get some challenges. In multicast routing a new approach was needed for loop avoidance. The large number of groups can be a critical issue as well. In contrast with Web and email traffic, the VoIP and the IPTV services are sensitive to delay and jitter. The importance of IPv6 and multicast is obvious. IPv6 came before the breakthrough, while multicast is now staring to conquer the inner networks of some autonomous systems.

2.1.5 P2P

The cheap and high bandwidth wireless access networks and the new generation of smart phones are generating new classes of interactions. As we mentioned above P2P video streaming (live and on-demand) has also seen a wide deployment and is being used by numerous users around the globe (7% of the total P2P traffic). In addition, there have been several proposals for designing peer-assisted content distribution networks (CDNs). In all of these systems, a receiving peer needs to be matched with multiple sending peers, because peers have limited capacity and reliability.

The network providers and the Internet service providers are facing new challenges both from a technical view and business model view. The widespread use of P2P solutions [62] by customers and the huge amount of cross network traffic generated by the P2P applications are two well-known technological challenges. This is why the network providers and the Internet service providers do not like P2P applications, and in some cases they try to detect and shape the P2P traffic.

2.2 The network architecture

The IP network is not a monolithic entity, but is built from many interacting intermediate or end systems. In this section we will provide a top-to-bottom overview of the current IP networks.

2.2.1 The Internet

The Internet is the network of the networks. It is built from more than ten thousand autonomous systems (ASes) controlled by different legal entities. The ASes may be divided into two types: transit AS and stub AS. These ASes collaborate with each other in order to transfer data from one end of the system to the other. The transit ASes are able to transfer the traffic where the source and destination is not in that AS. The stub ASes do not do this. The collaboration is done on two layers: on the data plane they accept or reject the data coming from a peer AS, and based on their own policies, forward the data in the direction of the target AS. Besides this, they collaborate in maintaining the knowledge of the interconnections of the ASes. This is known as the signaling plane. The interconnection aspects among the ASes is done at peering points where they exchange both traffic and routing information. The so-called backbone of the Internet is formed from these public or private peering points and the ASes. The capability of the backbone depends on the bilateral or multilateral agreements of the peering parties. Some ASes support IPv6 or Multicast and some do not. In the majority of the peering points only the IPv4 and Unicast communication protocols are supported.

As we mentioned, there is no central organization behind the Internet and there is no dedicated backbone. No one knows the exact topology of the Internet, and the same is true for the traffic flows on the Internet: no one knows who the communicating parties on the Internet are in any given time frame. The ASes may have their own view of the global traffic, but it is only a partial view of the global network. It may be more important to study the correlations between the overlay networks (P2P) with the underlying network (IP and Internet). Given a single AS (even from the biggest one), is it possible to detect an overlay which is evenly spread out over the Internet? This is an important question if we would like to detect botnets. We will study this issue later in Chapter 7. In the next part we will examine ASes more deeply and discuss the issues associated with the intra AS.

2.2.2 Autonomous systems

From a technological point of view [55], a separate AS is needed for the legal entities who are willing and able to act as a transit network/transit AS. In practice, a legal entity may own an AS number and the visibility of this fact depends on the capabilities of this entity and the length of the prefix it owns. These intra AS networks may span the whole world, but these networks may also be located in a single place such as a campus. In most cases these networks are not ad-hoc, but they follow the well-known hierarchical engineering approach [105] where the network is divided up into the core, distribution and access layers. These layers have their own specific role in the network and the active device vendors design their device portfolios based on these three layers. In the following we will

discuss the roles of these layers.

Access layer

The goal of the access layer is to provide the last mile for the end systems. In the case of an Internet service provider (ISP), the end system stands for the point of presence (POP) at the customers location. This layer is the most intelligent and offers many services. Based on the classical engineering approach, most of the services requesting some state handling on the active device should be implemented in this layer. In most cases there are security-related issues like firewall and QoS related issues like traffic admission are handled in this layer. In a nutshell, the access layer is the place for stateful services and it can focus on a smaller region.

Distribution layer

In a larger region the islands of access layer networks are connected to each other through the distribution layer. The main goal of the distribution layer is to enforce the local routing policies and provide a redundant interconnection path for a set of access layer islands. The stateful services are less common in this layer.

Core layer

The goal of the core or the backbone is twofold: it should provide redundant data paths for the regional distribution layer islands, and it should handle the connection to remote networks (ASes). In most cases this layer is simple and does not have any stateful services.

2.2.3 Summary

We saw that the Internet is a network of autonomous systems which are networks of active devices organized in a hierarchical way. We mentioned the placement of the stateful services, but it reflects the past best practice and it may change in the future. One aspect of the so-called future Internet is to foster collaboration among the applications and the networks. The network should understand the applications and it should provide context-based services (e.g. routing for Web services [13]). One step in this direction is the active network idea [128]. In the next section we will examine the architecture of the active devices and their capabilities.

2.3 Active devices, architectures, capabilities

An active device in our terminology is a device which provides a service to the network. In most cases these are the devices working in the second and third layer of the OSI, but in some cases they can reach even the seventh layers of the OSI model. The performance and the capabilities of a given device heavily depends on the architecture of the device. In the following we will review the main hardware architecture solutions of the currently available active devices. As we focus mainly on the IP aspects in the thesis, below we will focus on the router architectures.

2.3.1 Architectures

The task of an IP router is to make decisions chiefly based on the destination address of the packet and to send it out on the given interface. From a functional point of view, the router has two main planes: the data plane and the control plane. The decisions are made with the help of the routing table, which is maintained by the control plane and it consists of a list of target networks and the next hop addresses. Based on the CIDR [130], the longest match is selected. There are many other tasks a router could or should do, but we will discuss some of the services requesting stateful operations in Section 2.4. Here we will overview the main architectural approaches. The processing of the packets could be done by a central unit or distributed by distributed processing units. The processing itself could be done by one or more processors or by an ASIC (Application Specific Integrated Circuit). The devices in the access layer mostly support centralized processing by a single processor; the distribution and core layer devices are mostly based on distributed processing and some of them have and ASIC for a selected set of services. A more detailed overview of the router architectures can be found in articles like [69] and [38]. For a new distributed approach the reader can browse the webpage mentioned in [76].

2.3.2 Support for stateful services

Stateful packet handling means that the packets are handled based on some internal state maintained by the router. A basic stateful service is the routing itself where the router consults the routing table and, based on the results of this lookup, it forwards the packet toward the destination. The processing power needed for this lookup could be quite high. For example in a distribution layer router with 48 interfaces each having 1 GBit/s data transfer capability, it means that in the worst case for small packets (e.g. 100 Byte long), the device should handle 64 packets every micro second. In other words, it has about 15 nanoseconds for each packet. The access speed of the currently available memory chips is in the range of 55 to 23 ns [11]. In order to be able to find a given entry, multiple

lookups are needed. Depending on the data structure, the number of memory accesses for exact match depends on the number of entries in the table (for a B-Tree data structure this is $O(log(n))$, where n is the number of entries). The state-of-the-art silver bullet for solving this issue is the CAM (Content Addressable Memory), which is a hardware implementation of the associative array. It returns the address of the cell (or the content associated with the cell) in one memory access cycle. For the longest matching lookup a special CAM called the Ternary CAM is used. The price and the power consumption of these memoriy chips is high [74]. Due of this, even in the high-end routers the storage capacity of the TCAMs is around several hundred thousand entries. This could cause a serious bottleneck [95].

2.3.3 Summary

We saw that there are various possible architectures available. The low-end routers solve the decision making process with the help of CPU and conventional memory while the middle- and high-end devices use TCAM for the lookup. TCAM is a scarce resource, so the number of entries needed to be stored is of crucial importance. In this chapter we provide a short overview of the currently deployed stateful services.

2.4 Stateful services

As we said above, stateful packet handling in our terminology means that the packets are handled based on some internal state maintained by the router. One basic service in this portfolio is the routing itself.

2.4.1 Routing

In the CIDR-based routing, a decision is made based on the destination address of the packet. The router maintains a data structure called the Forwarding Information Base (FIB) [129], which is a search optimized data structure made from the Routing Information Base (RIB) that is based on the information coming from routing protocols and the link layer adjacency information. The size of this data structure depends on the where the router is. For an access layer device there could be several dozen entries. In the case of distribution layer devices there could several hundred or several thousand entries. With core devices connecting to the peering point there could be one or two hundred thousand entries. The content of this table is well managed with the help of different aggregation policies and it is stable.

2.4.2 Network Address Translation

This service is described in RFC1631 [64]. In a sentence, the NAT means that for a given packet we replace the source or destination address with another address from an address pool. (with PAT we do the same but for the ports). For the incoming packets we make the same replacement but in the opposite direction (i.e. we restore the original address). In order to know what was the new address – old address assignment, the router maintains a table called the address translation table. This table is consulted every time an incoming or outgoing packet should be processed. The number of entries in the table depends on the number of flows going through the device. In this case a flow is identified with the source/destination IP address and port pairs. As the number of flows in the backbone could be quite large it is rare to find this service in the core. But there are some exceptions; for example, some 3G providers provide private addresses to the customers and the NAT function is implemented in the GGSN [6], or in the case of some large ISPs a few million end users are behind the central NAT. In this case the number of flows is a critical issue. We will study this later with the help of real measurement data in Chapter 4 .

2.4.3 Monitoring the network

Monitoring the network is of critical importance for the network operators. The information coming from different network monitoring solutions is the basis for security, network planning, traffic engineering and numerous other areas. In the TCP/IP world, the SNMP-based counter reading is the most widely used information source. It is not resource intensive, so the network administrators can apply it even on resource poor devices without any deliberation. The type and the granularity of the information which a traffic counter can provide is not sufficient for a wide range of security, resource planning or other applications . The next level of abstraction related to the status of the traffic is flow-based accounting. The IETF standard Netflow [17] does this, providing a framework for identifying the flow and for generating statistics for a given flow. In most cases the flows are identified with the help of source/destination IP addresses/ports and the L3 protocol, but depending on the Netflow version other fields of the IP packet can be selected as the flow denominator. Statistical data about each flow is collected in the Netflow entries in the memory (the so-called Netflow cache) of the active device. These entries are updated (or if they are not present then they are inserted) upon the arrival of a packet which is member of the given flow. The entry contains the identifier for the flow (in most cases the previously mentioned fields) and the fields containing the generated statistics (e.g. the number of packets and aggregate bits of a given field). In order to avoid any overflow of the cache there are different mechanisms for purging the old entries. In the case of Cisco devices these are the following:

- Flows which have been idle for a specified time are expired and removed from the cache (the default time is 15 sec)

- Long-lived flows are declared out of date and removed from the cache (flows are not allowed to live more than 30 minutes by default and the underlying packet conversation remains undisturbed)

- As the cache becomes full, a number of heuristics are applied to aggressively age groups of flows simultaneously.

- TCP connections which have reached the end of a byte stream (FIN) or which have been reset (RST) will be declared out of date.

The data from the flow cache is exported to the flow collector (in most cases a PC) periodically based on the flow timers. This information is sent in UDP Netflow datagrams, each containing information about 24 to 30 flows. In the optimal case the monitoring traffic is about 1.5% of the monitored traffic [17]. In order to further decrease the amount of information sent from the router on some active devices there is a second level cache for aggregation. In this case the information from flows is aggregated first and only the aggregated information is sent through the wire. With this feature the granularity of the information available on the monitoring server decreases significantly.

Sampling

One method used to decrease the load on the processor is that of packet sampling. A white paper [123] says that with a 1:100 random packet sampling the usage of the processor was decreased by 75%. There are different sampling strategies (random packet/flow sampling, deterministic sampling), but one common feature of these solutions is the loss of granularity. In some cases (e.g. security) this cannot be tolerated. The authors of [29] provide a good overview of the open issues associated with packet sampling. They say that

- during flooding attacks router memory and network bandwidth consumed by flow records can increase beyond what is available;

- selecting the right static sampling rate is difficult because no single rate gives the right tradeoff of memory use versus accuracy for all traffic mixes;

- the heuristics routers use to decide when a flow is reported are a poor match to most applications that work with time bins;

- it is impossible to estimate without bias the number of active flows for aggregates with non-TCP traffic.

The authors of [29] offer a software-based solution which gives a better solution for the first three issues, but for the fourth one only a hardware-based solution is suggested. In short, the sampling can decrease the CPU utilization, but in this case some flows will go outside the visibility of the network administrator. The authors of [28] present a distributed infrastructure for handling the Netflow information. Scalability is achieved by sampling on different levels. Due of the employment of a wide range of sampling solutions, this approach is capable only of estimating the usage of traffic class properties of the network traffic. It cannot be applied for monitoring the security issues. The authors of [34] conclude that it is inevitable that systematic sampling can no longer provide a realistic picture of the traffic profile present on Internet links. The emergence of applications such as video on-demand, file sharing, streaming applications and even on-line data processing packages prevents the routers from reporting an optimal measure of the traffic traversing them. In the inversion process, it is a mistake to assume that the inversion of statistics by multiplying by the sampling rate is an indication of even the first order statistics such as packet rates. In summary, we can say that the feasibility of the use of sampling depends on the goal of monitoring. For traffic engineering it could provide enough granularity, while for the security issues (e.g. botnet detection) granularity and precision are not adequate.

Netflow placement

The choice of the monitored places depends on the goal of monitoring the capability of the devices and the topology of the network. A rule of thumb is that monitoring should be done on the edges of the network. However in some cases even the core devices should provide some kind of packet classification different from simple routing. The owner of the network should know what happens on his network [125]. This knowledge is necessary for both the traffic engineering and for security decision making policy. The ISP also should be able to influence the traffic on his network. This is necessary again for both traffic engineering and security tasks. In most cases in order to fulfil these tasks the active devices should classify the packet. A case study from Cisco [124] describes the motivation and the placement of monitoring functionality. They placed the monitors mostly on the WAN and the edge links, but they also monitored the different extranet and VPN traffic (it could be also regarded as one type of edge). The goal of the authors of [135] was to optimize the deployment of monitoring functionality on a real network. The objective of the monitoring was to have flow- or packet level-information about the traffic traversing the network. The cost function was defined with the help of the amount of capital investment. The network studied was built from Cisco GSR and

7500 routers. They found that the price of achieving the 100% coverage is double the price for 95% coverage. As the C7500 routers are now available for flow monitoring (this depends on the number of flows), they are put in places where there is less traffic and the upgrade of the GSR devices provided the most significant factor in the total cost of the upgrade. The authors of the article did not take into the account the total cost of ownership (e.g. with changing network conditions the line cards of the 7500 would not be appropriate), and they modelled only the price of different cards. In summary, the placement of monitoring functionality depends on the traffic and the network topology as well as the type and capability of the active devices. A change in the traffic profile could have serious consequences on the monitoring capability of the network provider.

Discussion of the resource consumption of the traffic monitoring

The use of the Netflow framework on a network device has its price in memory, processor and network bandwidth consumption. NetFlow performance impact comes mainly from the characterization of the flow information in the NetFlow cache and the formation of the NetFlow export packet and the export process. The high-end devices use TCAM and network processors for packet classification. In this case the performance does not depend on the number of flows, but the size of the TCAM memory limits the number of flows. The range of the number of flow supported is from 125 KFlows to 2 MFlows. Low-end devices use the traditional memory and the CPU for this task. The flow export is done in most cases by the processor and in a few cases by ASIC. A detailed analysis can be found in [123], where there is a description of a big scale experiment of the performance impact of the netflow on different Cisco router types (centralized/distributed, etc). The study said that as the number of flows increased, the delta between the baseline and NetFlow-enabled CPU utilization widened. In other words, the more IP flows are present, the more system resources NetFlow are required. The more active flows NetFlow maintainined in its cache, the larger the cache becomes and the more CPU it requires to sort through the cache. Another important aspect of network monitoring is the amount of data to be transferred through the network and processed on the network monitoring infrastructure. In the optimal case the monitoring traffic is about 1.5% of the monitored traffic. This ratio depends heavily on the traffic mix. Therefore we may conclude that the number of flows has a serious impact on both the monitoring capability and the traffic, and also that the amount of data generated by traffic monitoring and the placement of monitoring functionality depends on the traffic, the network topology, the type of and capability of the active devices. A change in the traffic profile could have serious consequences on the monitoring capability of the network provider. We will study the capabilities of active devices and traditional PCs later on in Chapter 4.

2.4.4 Firewall

The task of the firewall is to enforce the rules defined by the network administrator on the traffic going through the active device. There are two main types of firewall functionalities: the first is the stateless firewall where the decision depends only on a given packet and the the communication history is not taken into the account; while the second type is the stateful firewall where the communication history is important for decision making. Here communication history in most cases means the state of the flow. In this case we have the same scalability issues as we saw in the case of NAT. The placement of the firewall functions and the type of the firewall functions are both network engineering issues. The classic approach is that stateful firewalling should be done in the access layer. The rules in a firewall are described as a set of Access Controll List's (ACL). In most cases these rules specify the traffic with the help of IP port tuples. It is now clear for the network administrators that the source/destination IP/port tuple is not enough for application identification as the applications may use arbitrary ports. In the next section we will discuss this issue.

2.4.5 Application identification

From a security, QoS or traffic engineering viewpoint, application identification is becoming increasingly important. It is now a hot topic for the research community. As this is not the focal point here we will discuss only one solution called Network Based Application Recognition (NBAR) [127], which is a proprietary solution but it is available on almost every Cisco device. Other areas like botnet detection will be discussed in the next section. As NBAR is a closed solution, there is no exact description of the approach it uses to detect the application footprint, but some researchers suggest[39] that it should use some sort of deep packet inspection (DIP) with string matching. The scalability of this service has been studied by Cisco [126], but they have studied only the impact of the raw bandwidth to be monitored and they have not measured the impact of the number of flows. As this service needs the same bookkeeping as we saw in the case of NAT, stateful Firewall, and Netflow, it has the same scalability issue with an increasing number of flows. With encrypted traffic, DIP is becoming less precise but botnets are becoming increasingly sophisticated. We will study this issue in the next section.

2.4.6 P2P botnet detection

In recent years peer-to-peer (P2P) technology has been adopted by botnets as a fault tolerant and scalable communication medium for self-organization and survival [40, 32]. Examples include the Storm botnet [40] and the C variant of the Conficker worm [98]. The detection and filtering of P2P networks presents a considerable challenge [49]. In addition, it has been pointed out by Stern [118]

that the potential threat posed by Internet-based malware would be even more challenging if worms and bots operated in a "stealth mode", avoiding excessive traffic and other visible behaviour.

P2P botnets are a challenge to the security of the Internet and their potential threat should not be underestimated. Considering the fact that P2P botnets have not even begun to fully utilize the increasingly advanced P2P techniques to their advantage, the future seems even more challenging.

State-of-the-art approaches for detecting P2P botnets rely on considerable human effort: a specimen of the P2P bot needs to be captured and reverse engineered, message exchange patterns and signatures need to be extracted, the network needs to be crawled using tailor-made software, its indexing mechanism poisoned, or otherwise infiltrated, and a countless number of creative techniques have to be applied such as visualizations, identifying abnormal patterns in DNS or blacklist lookups, understanding propagation mechanisms, and so on (see [104, 40, 32]). Besides this, aggressive infiltration and crawling introduces lateral damage: it changes the measured object itself quite significantly [65].

While creative and knowledge-intensive approaches are obviously useful, it would be nice to be able to detect and map botnets *automatically*, and as *generically* as possible. Ideally, network monitoring and filtering tools installed at routers and other network components should take care of the job with very little human intervention based on the (often enormous volumes of) Internet traffic constantly flowing through them.

This automation problem has been addressed in the context of IRC-based botnets [121] and, recently, also in the context of detecting generic botnet activity [33] and, specifically, P2P traffic [50].

2.5 Summary

Here we provided an overview of the actual architecture of the IP network and the current trends shaping the future of the architecture and its applications. It is clear that the network is becoming even more intelligent, but even the actual semi-intelligent functions have their serious scalability issues. The effect of the number of flows on stateful services is known, but it has been studied only in a few areas without yielding an overall figure. In the background the memory access speed is the limiting factor; with the help of TCAM and memory banks the current high-end devices solve this poblem, but the size of the TCAMs provides a strict limit on the number of flows a device can handle. On other hand, we saw a wide range of the currently deployed stateful services. In the next three chapters we will study the impact of the number of unicast or multicast flows on the infrastructure by making quantitative measurements.

3

The impact of the number of multicast flows on the infrastructure

As we saw in Chapter 2, there has been a significant increase in the number of broadband users and with the advent of the triple and quad play services, the next killer application could be IPTV. We also saw in this chapter that the multicast service has no worldwide backbone but with the spread of IPTV it could conquer the access and distribution layers of the ISP networks. Scoping is a critical issue in the case of multicast networks as it is one possible tool for traffic engineering. Due of the address shortage, scoping is not straightforward with IPv4, so IPv6 could provide a viable solution. With efficient bandwidth usage we also get some challenges. In multicast routing a new approach was needed for loop avoidance. The large number of groups can be a critical issue as well. In contrast with Web and email traffic, the VoIP and IPTV services are sensitive to delay and jitter. The network operators should audit their networks to see how they can and should cope with the new challenges. The frequent testing of a network may provide administrators with some useful data and experience on making preparations for special situations that may arise. In this part we will present a general purpose framework for network measurement and our results in the area of IPv6 multicast scalability.

3.1 Related work

A popular approach in network testing is one of using traffic generators to model flows. There are many interesting applications for traffic generation, but they are very simple approaches or they are not maintained. One of the best known freely available traffic generators is the package described in [4], which supplies the user with a distributed testing capability. In its current state it is a miscellaneous collection of utilities. The distributed control of the agents is achieved with the help of a propriety protocol. The agents listen in on a specified port for instructions. One may write and implement software to control them. It supports many protocols (CP, UDP, ICMP, DNS, Telnet, VoIP (G.711, G.723, G.729, Voice Activity Detection, and Compressed RTP)). One advantage of this solution is the support of different probabilistic distributions for modelling different traffic scenarios. It also supports IPv6. The software package is written in C++ and it has been ported to both Linux and Windows. One can if one wishes use a Java-based GUI for managing a single agent. Compared to our approach where the user has the freedom to construct arbitrary packets, this one just has a fixed set of supported protocols. Our approach provides a message sequence chart editor where the user can specify arbitrary sequences and the task of synchronizing the participants is the task of the server. In D-ITG the distributed testing scenarios may be defined in configuration files (without synchronization) or they may be managed from a remote controller, but currently there is no tool comparable to our MSC editor for orchestrating different distributed traffic situations. We have not found any information about the support for IPv6 multicast testing on the Net. The only suitable one we found was the software package described in [112]. It was the only one available for this purpose. Although it is a very useful tool, it lacks a number of important features like membership testing and multipoint-to-multipoint testing. One can manually create arbitrary configuration files, but in this case the system administrator should do the work. It may be the best tool for a simple multicast network testing procedure where we are not actually interested in different traffic scenarios, but just want to know whether the network works or not.

3.2 Our solution

Our goal was to design and implement a general platform for network testing and protocol validation. To achieve this goal we set the following criteria for our framework:

- The user can define every bit of information of the sent and received packets.
- The user can define arbitrary sequences from previously defined set of messages.
- The user can define arbitrary scheduling for incoming and outgoing messages.

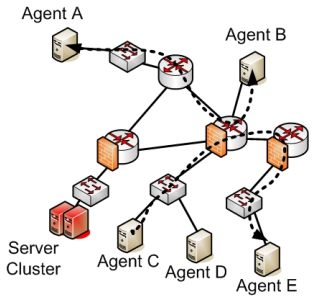


Figure 3.1: Infrastructure

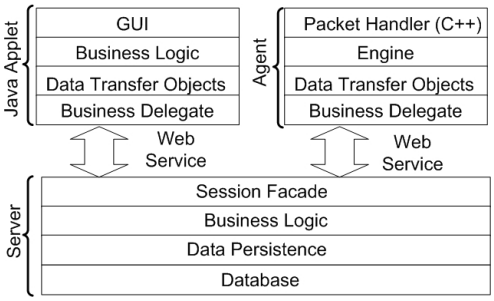


Figure 3.2: Architecture

- The user can define a distributed scenario where there are several traffic sources and destinations are arbitrarily located on the network.
- The system should be easy to use (i.e. user friendly).
- To reduce the burden of looking after a distributed system, it should be managed from one central point.

With this functionality we can not only test a system, but we can also validate and check the conformance of different protocol implementations.

3.3 Architecture

To fulfil the above criteria we opted for a centralized solution. As the reader will notice in the figure, there is a central server and an arbitrary number of agents.

The agents have an independent ability to execute the scenarios defined by the central server. They are the source and the destination of network traffic, and they may be the sampling points too. In the central point of our framework there is a server where the user can orchestrate different traffic scenarios. As we may like to provide access to our system from different locations, and which may be separated by firewalls, we opted for a Web-based user interface. Due to special user interface requirements we implemented the interface as a Java Applet (see Figure 3.2).

In spite of the effectiveness of multicast communication, we decided to use unicast communication between the agents and the central server because of its simplicity and firewall friendliness. The agents may be placed on network segments that are separated from the central server by firewalls; hence we used Web services as a communication channel between the central server and the agents. As we would like to test the network, it may happen that there is no connection between the server and one or more agents. We found a solution for this problem in the DBeacon software package where

there is no central point and the whole system is built as a peer-to-peer solution. But owing to its complexity and unpredictable nature we later decided to reject this solution. To overcome the network failure between the server and the agent one can manually copy the scenario file to the failing agent. We do not require special purpose or dedicated machines for an Agent role. As they may function as normal desktops due to security constraints, it is not a good idea if they act as servers. So the communication is effectively one way. The agents can access the central server, but the central server cannot initiate communication. To ensure the manageability of the agents they are connected to the central server by a given schedule. The defining of this schedule is the task of the central server. For some measurements, scheduling is critical. Suppose, for instance, we would like to measure the delay between the sending and the receiving of a multicast RTP packet. As the clocks of the agent machines may not have been synchronized properly, we cannot rely on them. But we can provide two solutions for this problem. An offline solution is one where the agent sends its local clock value to the central server during the to-do list download. The central server modifies the scheduling based on the difference between its clock and the agent's clock. This solution can be used in most situations, but when precise scheduling is needed and different clock speeds are not tolerated an online solution may be used. The agents connect to a special scheduler procedure which returns a value when all the agents have been connected and the clock on the central server hits a given value. The central server could be a single point of failure, but as we would like to use this system for the continuous testing and monitoring of a network a failure of the system cannot be tolerated. Hence we designed and implemented a multilayer approach whose diagram is shown in Figure 3.2. Both the database layer and the business logic may be clustered. The logic is implemented as EJB 3.0 session beans. Some of them just have a Web Service interface for the agents and controlling Applet. We used POJOs to represent the data. The persistence of these objects was handled by the Application server.

3.4 Services

Now we would like to describe the services provided by our framework and the way they were implemented by us.

3.4.1 Network handling

As the Java language is a high level language and the development cycle is shorter than that for an unmanaged environment, we implemented the client in this environment. The biggest challenge for us was raw network handling. The Java platform provides only high level network handling, beginning with its capability for socket handling. As we would like to give the user the chance to define an

arbitrary packet we extended the capabilities of the Java platform with a new API to handle raw network traffic. We implemented this functionality in C++ and ported it to the Linux and Windows platform. With this API one can send MLDv2[132] packets from a Windows box that does not have the capacity to handle MLDv2 packets, or one can send PIM-SM[30] Hello messages from a machine which is not a router. The Java RTP stack can send IPv6 RTP packets only with a unicast source and destination addresses that have DNS entries. In some cases this is not available. With our solution the user can define RTP packets and handle them without relying on a DNS service.

3.4.2 Agents

The agents are installed on different machines in different parts of the network, independently of the number of firewalls between the agents and the central server. The first task of the agent during the start-up procedure is to register itself on the central server. During this process the agent transfers all of its special properties to the server like the number of interfaces and the defined IP addresses. This data is refreshed only when needed. The user may group the agents and define specific properties for them (e.g. message sequences).

3.4.3 Templates

The freedom to define arbitrary messages is not of much value without an easy-to-use toolset. No one will define a message sequence one bit a time and calculate the checksums as well. Hence we designed and implemented a powerful template engine for this. The templates have the following properties:

- Inheritance
- Composition
- Auto fields
- Alias handling

With the help of inheritance one can define message families from less specific to the most specific messages e.g. IPv6 packet, IPv6 packet with UDP encapsulation, or an IPv6 packet with a UDP or RTP encapsulation. With the help of composition we can achieve the same results. With these solutions one can define message libraries and reuse them. And using auto fields one can define the content of a field to be filled by the GUI. The checksum is a good example where the user may select the fields from which the checksum is to be calculated. The user may define friendly aliases and use

them in the GUI instead of the long IPv6 addresses. Another example is when the user would like to set up a large message sequence and the difference between the preceding and subsequent message field can be defined as a logical expression. With these features a time-consuming test case setup may be less monotonous for the user and be less error prone.

3.4.4 Sequence definition

To describe the message sequences we constructed an easy-to-understand XML syntax based on the ITU-T. Z.120 [52] message sequence chart recommendation. We then selected the most interesting subset of the functionality defined in Z.120 for the implementation. With the help of the GUI (shown in Figure 3.2) the user can define sequences for an arbitrary number of agents. These sequences are then stored in the database. When an agent downloads its own sequence, the server creates a customized sequence with synchronization and collects the messages from the general sequence that are of interest to an agent. In this way the user is able to create complex scenarios and the agents will just receive the communication sequences they are involved in.

3.4.5 Probabilistic functions

We applied several well-known probabilistic distributions that are used in the telecommunication and traffic modelling fields. One can define the value of an auto field as an output of a probabilistic function.

3.4.6 Reporting

The user can define the interesting properties to measure during a test. This might be measureable traffic parameters like delay, jitter or the difference between the defined and the received message sequence. The result might be the whole received message sequence (without data). The results of a measurement are then transferred to the central server after the measurement has been taken. On the server side one can use a visualization framework to analyze these results.

3.5 Measurements

For a system administrator to guarantee the continuous operation of the managed network, a good knowledge of the capabilities of the network is required. One common solution used by most administrators is to monitor the network with the help of an SNMP-based software package. This solution may provide some knowledge about the actual state of the network, but it cannot provide much information about the effects of planned or unplanned special events on the network. For example,

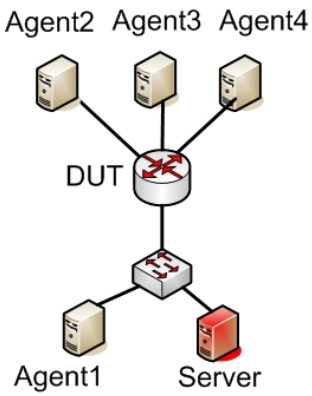


Figure 3.3: The setup

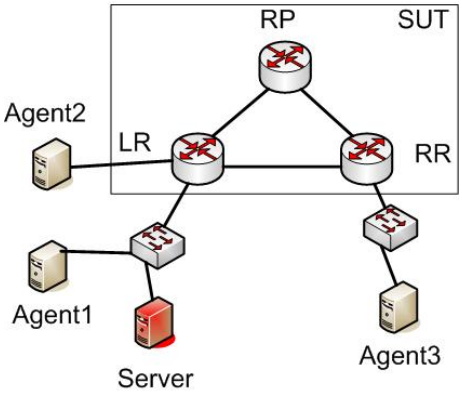


Figure 3.4: The DUT and the agents

whether a company has decided to migrate the voice communication from the POTS to a VoIP solution based on the current network. Due to the undetermined nature of the network traffic, the complexity of the network and lack of detailed documentation about the capabilities of networking devices, the analytic approach for predicting the possible impact of the new network traffic in most cases cannot be used. A more popular and useable approach is to measure the network in different scenarios. Currently there are only basic devices available for this task. Most traffic generators can only be used with fixed configurations and as they intended to be desktop applications they are not meant to be used as distributed applications. The recommendations for system testing are mostly based on stress tests. We think that knowledge of the behaviour of the managed network in an everyday situation could be more important than during peak periods. In spite of well-known theoretical models for various types of traffic, we were not able to find any suggestions about the kind of measurements we should make.

3.5.1 IPv6 multicast measurements

Our original goal was to test the capabilities of the Linux IPv6 multicast router, especially the PIM-SM implementation. RFC 3918 [120] describes the methodology of IPv4 multicast testing and RFC 2432 [27] describes the terminology used in this area. These documents only specify a single source multiple receiver testing scenario. A draft we found [97] contains several additions to the benchmarking methodology which can be interesting for IPv6 benchmarking. Below we will present our results for IPv6 multicast group capacity and join delay in different traffic scenarios and network topologies.

A	Processor	Mem.(MByte)	Net. card(100MBit/s)
RP (Rand. P.)	P4 1300 MHz	512	2
RL	P4 1300 MHz	512	4
RR	Cel. 600 MHz	256	3
Agent1	P4 1300 MHz	512	1
Agent2	Cel. 600 MHz	256	1
Agent3	Cel. 600 MHz	256	1

Table 3.1: The hardware environment

N.Ch	64	512	1500
10	50000	50000	49200
100	49514	49664	43311
1000	46813	43808	41642
10000	n.a	n.a	n.a
60000	n.a	n.a	n.a

((a)) Packet loss

N.Ch	64	512	1500
10	17	23	14
100	227	254	319
1000	3800	3700	4200
10000	72777	>70000	>70000
60000	>70000	>70000	>70000

((b)) Delay

Table 3.2: Results

3.5.2 The configuration used

We set up a sample configuration shown in figures 3.3 and 3.4 with Linux IPv6 PIM-SM routers and Linux-based clients for them. The machines had the following configuration: Debian Sarge,MRD6 0.9.5 PIM-SM [111] implementation, Zebra Ripng as a unicast routing algorithm, Java 1.5. Above Table 3.1 lists the hardware specifications of the machines.

3.5.3 The number of supported channels

In the experiments our goal was to learn more about the dependence between the number of channels and the packet loss rate. Our tests were done with an equal number of packets (50000). To test the traffic we used an IPv6-based UDP packet of variable length and fixed content. The only varying parameter in the UDP was a serial value. On the receiver side, the received serials were the result. Each MLDv2 packets contained 50 multicast addresses with an exclude directive. We conducted the measurements for both topologies (SUT and DUT, figures 3.3 and 3.4). In both cases the traffic source was Agent3 and the traffic destination was Agent1. The number of received packets is shown in *a* subtable of Table 3.2.

Evaluation

The system worked well up to 100 channels. With 1000 the packet loss rate increased, but only to about 2-10%. With a larger packet it was greater. If we injected the same traffic several times the packet loss rate decreased by 1-5%. We suppose the reason for this behaviour can be found in the FIB implementation. When we chose 10000 channels or more the system could not cope with it. The RR router processed about 4300 subscriptions and from these subscriptions only 3150 were registered on LR. We slowed down the subscription rate, but the best result we were able to achieve was that of registering 5600 channels on RR and 2947 channels on LR. It was surprising to us that the RR started sending PIM-SM Join messages only after processing the majority of the MLDv2 Register messages, rather than in parallel. It seems that the MLDv2 handling task has a higher priority than the PIM-SM signalling task. The multicast traffic for 10000 channels generated by Agent1 used about 60 MBit/s of bandwidth. Despite this low value the LR was totally overloaded during PIM-SM Register packet generation. From this experiment we may conclude that this system is well able to handle some 10-40 channels. Clearly the number of channels handled by the routers strongly affects the performance of a multicast network. A DoS attack on a multicast network aided by a large number of multicast channels can pose a real threat. The real network traffic is not significant (in the case of MLDv2 Join packets, several tens of ICMPv6 packets), but the impact of this traffic might be devastating. So we need safeguards.

3.5.4 The channel join delay

Here we measured the channel join delay for different channel numbers. We measured the time between the last MLDv2 packet and the first arriving UDP packet in milliseconds. The results that we obtained are listed in the *b* subtable of Table 3.2.

Evaluation

It seems that the delay is proportional to the number of channels. For larger packets the delay is bigger, but the difference is not significant.

3.6 Conclusions

In this chapter we presented our new network testing and protocol validation framework. The strength of this framework lies both in its user friendly GUI and the support it provides for defining a network traffic from top to bottom. As we mentioned previously, the current network testing scenarios are mostly concerned with benchmarking. We think that measuring a real network situation with a

lot of agents can provide the same or more valuable data than that obtained from benchmarking. The probabilistic approach where the traffic parameters are defined in terms of known probabilistic functions will add new data to the network testing field. Here we did not evaluate the protocol validation capability, but rather we measured the channel handling capabilities. But we think that the protocol validating capability should be widely used among network protocol implementers. During the testing phase it turned out that, based on RFCs, it is not a trivial task to fully specify a packet in detail. Hence we would like to define the most interesting protocols for our framework and we plan to make these sample configurations available on a community site. Here presented the results of our measurements of the channel handling capabilities of the MRD6 multicast routing daemon for Linux. In the literature we have not seen any such results for MRD6 or any for the IPv6 multicast routing solutions. In our experiments it turned out that the multicast network can be an easy target of a DoS attack. With a relatively small packet number, a multicast network can be shut down. In a real world scenario some rate limiting solution should be used.

4

Impact of the number of unicast flows on the network infrastructure

As we saw in the previous chapters, stateful services have their scalability constraints depending on the architecture of the active device (see Chapter 2). In Chapter 3 we investigated the effect of the multicast traffic on the PC based infrastructure. Now would like to study the effects of the number of unicast flows on the active devices.

4.1 Traffic generator

In order to simulate a high network load that would generate high flow numbers, a traffic generator was necessary with the capability to create and send specially crafted packets. The UDP packets generated with spoofed headers will cause the targeted router to register a given number of different flows (that is, to simulate a number of client to serve). To meet these needs we applied the framework described in Chapter 3. Here four parameters were used to tune it. The first parameter was the number of flows to simulate; that is, how many fake source addresses to generate. The second and third parameters were the destination IP and port range to send the packets to. The last parameter was the size of the packet the utility should create. When run, after parsing the parameters, the application performed from 1 to 10 million iteratios. In each of these iterations the algorithm picked a fake source address and the crafted packet was sent to the destination address and port.

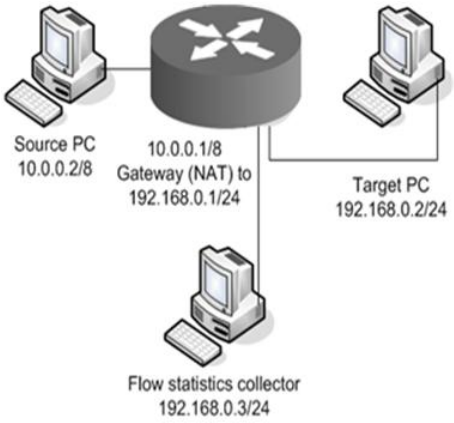


Figure 4.1: The setup

4.2 Environment

Figure 4.1 shows the test setup for the simulation: it consists of a machine as a traffic generator in a local subnet; a router that was either a PC or a Cisco router; a target machine and a monitoring machine in another subnet. The list of routers tested is shown in Table 4.1. The generator device had an IP from the network 10.0.0.0/8. The target machine had an IP from the network 192.168.0.0/24. The router would act as a gateway between these networks, either with NAT turned on or off depending on the test case. The source and target machines and the flow collector machine had the following common setup: a 3.0GHz Intel Pentium IV CPU, 1GB RAM and Realtek Gigabit Ethernet interface, all of them running Debian Linux 5.0. The PC router had two 2.4GHz Intel Xeon CPUs with Hyper Threading and with a 2.5GB RAM and two Gigabit Ethernet interfaces, running Debian Linux 5.0.

4.3 Test cases

We tested our network with 1000 byte-sized packets and for the following number flows (or virtual clients): 1 000, 10 000, 100 000, 1 000 000, 10 000 000; with three router settings: first with simple routing, second with NAT enabled and third with NAT and Flow export enabled. We monitored the CPU usage on the routers, and the number of dropped packets (by checking the number of packets arriving at the target PC). With simple routing the routers could only select the path in the network for each packet coming from the source PC and relay them to the target PC. This scenario uses the least amount of resources and shows how the routers react to high traffic load. With Network Address Translation enabled, the routers have to translate packet headers and track and maintain basic data about each active connection. This means extra CPU overheads and memory usage when

Type	Setup	Proc.	Mem.	TCAM	Traget layer / Description
2811	2 FastEthernet interfaces , 2 Serial interfaces,1 Virtual Private Network Module	Processor board ID FCZ12047254	256	-	Access layer – small to medium-sized businesses (that is, a top of 500 employees)
7600	RSP720-3C	PowerPC 1.2 GHz	1024	ACL 128K NETFLOW 128K FIB 256K	Distribution/Core layer – carrier-class edge router offering high-density Ethernet switching and routing with 10Gbps interfaces. It is designed for enterprises (more than 1000 employees))

Table 4.1: Devices tested

the number of flows grows. The third setup, NAT with flow export, enabled the routers to maintain flow statistics, aggregate this data and export this information in regular intervals to the flow collector PC. This not only generates extra CPU overheads and memory consumption, but also increases the bandwidth used when exporting the Netflow packets to the collector.

4.4 Conclusions

In most cases it seems that the bottleneck is the CPU . Maintaining the NAT table and translating the packet headers on-the-fly requires high computational capacity. The less advanced Cisco 2811 router with moderate capabilities is more likely to drop packets as the flow number increases, in contrast to the high-end Cisco 7600. The raw power of the PC router's CPU provided an advantage in performance compared to the Cisco routers. Without netflow monitoring, by increasing the flow numbers the number of packets lost increases proportionally. Enabling the netflow monitoring and export means measureable extra overheads in terms of CPU usage and network load. This results in higher packet loss and also hinders accurate netflow exports and statistics. With high flow numbers and high traffic load, it is quite likely that along with data loss there will be a loss in statistical data accuracy and problems with proper monitoring. With NAT turned off (simple routing) neither of the tested routers produced any packet loss. During the simple routing tests, the PC router and the Cisco devices both never reached maximum utilization.

Figure 4.2 shows the packet loss ratio for each router and setup combination. It is obvious that

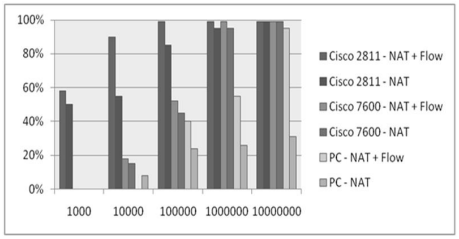


Figure 4.2: Packet loss

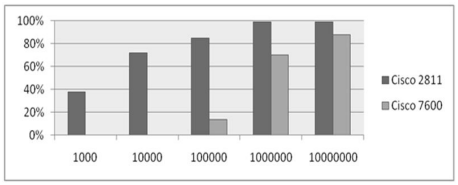


Figure 4.3: Netflow export loss

enabling the flow export will cause additional loss. Figure 4.3 shows how Netflow export accuracy is affected by an increase in number of flows. With higher flow numbers the ratio of Netflow export packets rises. Note as well that the Cisco 7600 router's CPU utilization does not reach 100% when packet dropping occurs. This is probably due to memory constraints or the size of the NAT table.

5 P2P network infrastructure

In order to counter the negative treatment it currently receives, the P2P community is working hard to design ISP friendly P2P protocols. The current emphasis of this research and development activity is on traffic localization. In this chapter we shall argue that the number of flows created by the P2P application should be considered when designing ISP friendly P2P protocols. There are studies which show that besides this, P2P protocols are responsible for a significant percentage of the total traffic volume: the so-called "elephant" and the P2P protocols are also responsible for a significant percentage of the generated small flows called "mice". Currently the effect of the high number of small flows is not well understood by the P2P community. In this chapter we wish to show that the P2P traffic is responsible for a significant number of flows on the backbone. After, we would like to provide a summary of the well-known ISP friendly approaches and point out that currently the number of flows is not considered in most studies.

5.1 P2P solutions

There are a huge number of design options for P2P overlays [81]. Here we will give an overview or bird's-eye-view of existing P2P overlay classes.

Unstructured P2P Overlays The broad class of *unstructured* overlays refers to random topologies with different degree distributions such as power-law networks [106] and uniform random

networks [61]. They offer no possibility for routing or key lookup, and they support flooding, random walk, or variations of these, as search methods. Gossip protocols are also well supported [67].

Superpeer Overlays In superpeer networks the peers are not equal: a small subset of the peers are automatically selected as temporary servers to help functions such as search and control [90]. Many P2P applications such as Skype, FastTrack and Gnutella [81] apply superpeers.

Structured P2P Overlays Structured overlays are distributed linked data structures designed for efficient routing (ID search). Nodes have unique IDs that can be used to address them. The overlay is organized to make efficient routing to a given ID possible. All versions of these overlays can be described as having a local structure based on some metric (often a ring, along which the IDs are ordered) and long-range links that serve as shortcuts. Shortcuts are typically arranged in a way that is consistent with the optimal arrangement described in [72].

5.2 Number of flows generated by the P2P applications

It is clear that the P2P ecosystem is responsible for a significant percentage of the total IP traffic. However, the flow level distribution of this traffic share is not well understood and has not yet been analyzed. As we saw in the Section 5.1, the different P2P approaches use different peering strategies and also a selection of the underlying communication services. Due of this heterogeneity it is not trivial to treat the P2P ecosystem as one class of applications from the viewpoint of the number of flows generated. This issue arises in [7] where the traffic generated by the World Wide Web (WWW) is compared with the traffic generated by the different P2P protocols at an Internet connection of the University of Calgary. In the client-based analysis, the authors found that while the number of concurrent flows for the WWW lies in the range of several tens but at most one hundred the number of concurrent flows for P2P applications is one magnitude higher. In a protocol-based analysis they found that Bittorrent users are responsible for this phenomenon as over 24% of the them have over 100 parallel connections. We may conclude that the P2P applications generate a significant flow volume. In the next section we will overview the actual focus of ISP friendly research and development.

5.3 ISP friendly P2P: state of the art

Most of the P2P networks deployed usually employ an arbitrary neighbour selection procedure. The result of this random neighbour selection procedure is a situation where some links of the overlay cross

multiple networks and continents [87][3]. It is common among the Tier 1 and Tier 2 network providers to calculate the fee of the peering based on the cross border traffic. As the P2P protocols generate significant cross border traffic, the Tier 2 and 3 providers should pay more to the Tier 1 providers. In order to cope with this issue, two main approaches have been considered in the literature. The first achieves locality with the help of selecting those peers with low latency. The second approach to achieve P2P-locality is to use ISP-provided topology information. In fact, the authors of [2] introduce an ISP operated oracle service for the P2P applications. The P2P node may send a set of alternative neighbours and the oracle sends back the best one for the ISP. The decision of the oracle about the metrics applied for decision making are not fixed, but they are in the hands of the given Internet service provider.

Another article [12] describes a solution using the already existing and deployed content distribution servers as the oracle. The basic idea is to construct a metric based on the localized DNS redirections applied by the CDNs. As the CDNs route the traffic according the ISPs local rules, the solution is AS agnostic. One issue associated with this solution is that the intra AS localization for smaller ASes is not supported. Another aspect is that the win-win model, which is the basis for the P2P-ISP ecosystem, is fulfilled only on the side of ISPs as there is only a small gain on the side of the P2P participants. This is mostly because of the weak access network connections. In [45] they describe an approach where the oracle information comes from the BGP tables, the geolocation services and the BGP updates. Based on this information they construct an "inter-ISP" topology and "intra-ISP" topology based on the estimated point of presence structure and distances among the point of presence sites. By combining these two sets of information they create an Internet level point of presence map. The suggested decision-making point depends on the architecture of the given P2P solution. In the case of centralized protocols like Bittorent this could be applied to the tracker server; in the case of decentralized protocols like Gnutella it could be the task of the leaf nodes to select the best source from a list of possible sources. The algorithm is too CPU demanding to be placed on each decision-making point, so they suggest the deployment of an oracle server at the premises of the given ISPs. These oracle services could provide the necessary information for decision making. The metric applied for the validation of this solution was the physical distance and the AS level distance.

In [1] they extend the work described in [2] with an analysis of end-user experience. In the case of decision making it also involves taking the bandwidth of the last hop into account. They study the effect of the solution on the Gnutella P2P protocol on five different topology models. The authors of [41] give an overview of the motivation and the possible forms of ISP, end-user and overlay provider collaboration. It describes two forms of collaboration, namely the oracle service (SmoothIT Information Service (SIS)) provided by the ISP and the ISP-controlled peer participating in the overlay

(ISP-owned Peer(IoP)). The goal of these collaboration activities is to localize the traffic (with SIS and IoP) and stabilize the network (with the help IoP). The article also gives a good overview of the different rules of thumb agreement among the Tier1,2,3 players (e.g. 95% rule). It shows that for the 95% charging model depending on the P2P resource sharing strategy, in some cases the locality of the traffic has only a small effect on the cost of the ISP. As the charging is based on the difference between the incoming and outgoing traffic in the case of tit-for-tat solutions the traffic in both directions will drop and have only a small effect on the difference between the incoming and outgoing traffic.

The usability of the locality also depends on the spread of the resource. If there are no alternatives in the network of the ISP then there is no chance for localization. Articles [113] and [87] describe the project ALTO (maintained by the IETF) approach which tries to standardize the oracle and the open issues in this area such as privacy and security considerations. It takes into account the Proxidor [3], P4P [134], and H12 [68] protocols as potential candidates for implementing the oracle protocol. The authors of [116] address the scalability issues of the oracle for unstructured P2P networks. They suggest using a gossip-based extension of the oracle to ease the burden of the oracle server. The authors of [89] conducted measurements on the Bittorent ecosystem in order to evaluate the effects and the effectiveness of localization. They found that in most cases for the client localization is not feasible as there are too few peers inside a given ISP and the quality of the service perceived by the P2P nodes could also degrade with increased localization.

Another important finding is that download depart behaviour limits the chance of localization too. The authors of [108] present a measurement-based analysis of the feasibility of the localization of the Bittorent protocol. With the help of in-depth and large datasets collected from the tracker server and the gossip protocol implemented in the Bittorent protocol for collecting all the IP addresses of the participating nodes, it evaluates two localization approaches on the dataset; *Locality only if faster* and *Locality*. They found that, depending on the localization approach, about 10% to 55% of the traffic could be localized. In the case of random peer selection this value ranges from 1% to 10%. The second conclusion is that the non-localizable Torrents where there are not enough seeding partners inside the ISP provide the upper bound to localization. The authors of [117] evaluate the effectiveness of the Vivaldi [22] Internet Coordinate System. The ICS solution could act as an oracle to help the peers in the localization process. Vivaldi calculates the network coordinates based on RTT measurements with the help of a spring relaxation problem. They found that because of the queuing delay of the ADSL links, Vivaldi cannot provide an accurate network coordinate system (in terms of adjacent geographical or AS distance). In summary:

- The problems of the ISP caused by P2P users depends on the contract type. If it is based

on the difference between the incoming and outgoing traffic then localization may have little effect on the cost of the peering.

- RTT as a metric could be applied with significant gains, but it does not take into account the traffic engineering decisions of the ISPs.
- Localization is strongly bounded by the demographics of the given P2P solution.
- Adopting the AS hop count as a metric for localization could also provide new results, but it is not optimal for ISPs covering multiple ASes
- The only metric employed previously of this was the volume of the traffic; the TCO of a network and the impact of the P2P traffic on the TCO was considered based on the traffic volume.
- The usability of localization (from an end-user perspective) decreases as smaller and smaller parts of the networks are considered. This is because of the given demographics of the P2P network. In a smaller network the chance of having peers with the same downloading goal tends to be small.

5.4 Conclusions

We saw that some P2P protocols are responsible for a significant number of flows initiated by a client. The state-of-the-art ISP Friendly P2P research community focuses on the traffic volume and the localization of the traffic. The number of flows as an additive metric for optimization has not yet been considered. Based on the measurements presented in Chapter 4, we think that the number of flows generated by a P2P application is important and this metric ought to be included in current aims of P2P optimization efforts.

6 Conclusions

Summary: The impact of the number of flows on the performance of the stateful services of the active devices has not been studied. Based on our measurements we showed that the number of unicast or multicast flows has an important impact on the infrastructure and these aspects should be considered in current and future network planning and application development.

Theses:

Thesis 1 *The performance of stateful services in the distribution and core layers depends heavily on the number of unicast or multicast flows.*

Thesis 2 *The ISP friendly P2P must take into account the number of flows generated by the overlay too.*

The results shown in this thesis group are all the results of the author. The results related to the multicast traffic of this contribution point were published in research paper [Bil06].

In the next part we would like to answer questions arising from this part:

1. As both the Internet and the botnet on the top of the Internet are distributed in nature, it is interesting to ask whether it is possible to detect a sophisticated botnet from a single point (single AS). We will study this in Chapter 7.

2. As we saw in this chapter, the degree of the nodes forming the P2P overlay is important from the point of view of network scalability and of hiding. It is an interesting question of whether it is possible to build a low degree overlay which is stable even in the case of significant churn. We will provide a solution for this problem in Chapter 8.

3. We saw that with the all-IP solution, we use the same protocol stack on the top of different access technologies. In the case of 3G, the bandwidth is a scarce resource and compression seems to be a feasible solution for protocol adoption. One may ask whether the current compression algorithms can be applied without modification. We shall present an in-depth study of SIP compression in Chapter 9.

4. If we consider a localized but extremely distributed system with strict performance criteria (file sharing), a question that might arise is how we could achieve robust, but effective consistency in the case of a file store. We will provide a solution to this in Chapter 10.

Part II

The Applications

7 Hiding botnets

In Chapter 2 we discussed the threat posed by the P2P botnets. In this chapter we examine how techniques presented in [50] perform in the case of botnets. Instead of looking at real traffic traces, we will base our methodology on simulation: we define synthetic flows on top of an AS-level model of the Internet assuming various P2P botnets. This is necessary because our main goal is not to evaluate current botnets, but rather to explore some advanced P2P techniques such as localization and clustering that *future* P2P botnets are likely to adopt to avoid detection.

Our main contribution is demonstrating that P2P botnets can easily hide their traffic even at the largest backbone router if they apply a few P2P techniques that are available from the literature or that are fairly evident, while being able to maintain their overlay and hence their malicious activity as well.

The implication is that local and isolated efforts are not very promising; if we would like to protect the Internet from P2P botnets that are likely to increase in size and sophistication, and that are still very far from reaching their full potential, we need to fight fire with fire and start to devote serious efforts to the consideration of P2P infrastructures and algorithms for automated detection.

7.1 Focusing on Overlay-Related Traffic

If we want to identify and filter P2P botnet traffic in an automated way, we can focus on roughly three kinds of activity: propagation, attacks by the botnet, and overlay traffic, which involves repairing failed

overlay links, adding new nodes to the overlay, and also applying spreading and searching commands of the botmaster.

We argue that the most promising approach is to focus on overlay traffic. It has a small volume, but it is arguably the most regular and reliable traffic that any P2P botnet generates, since the overlay network has to be constantly repaired, and bots need regular information about commands of the botmaster, updates, and so on.

Although the propagation of bots can be detected if it involves port scanning and similar suspicious network activity (see [10], for example), bots can also spread under the radar via email, websites, file sharing networks, ad hoc wireless networks, or even via the old-fashioned way by infecting files on pen drives and on other portable media that generate no network traffic at all [71, 133].

Certain "visible" attacks—such as DDos, spamming or brute force login—could also be detected automatically. In an optimistic scenario, we can immediately identify and block those bots that participate in these attacks. But this is still far from being enough: P2P overlays can apply very cheap and simple methods that can enable them to tolerate attacks extremely well even if large portions (even 80%) of the overlay gets knocked out [61]. In addition, certain types of malicious activity, such as collecting personal data (bank account information, passwords, etc) can blend into (even piggyback) overlay traffic perfectly.

One interesting idea is that—instead of concentrating just on the overlay or only on attacks—we should look for a *correlation* between groups of nodes that produce a similar flow-level behaviour due to overlay traffic and groups that perform attacks, as proposed in [33]. However, as acknowledged in [33], such correlations could be reduced to a minimum by a sophisticated botnet.

The basic motivation of our work is that we think that the *structure* of P2P networks is a very promising, although difficult, target to try to detect. The structure is completely insensitive to actual flow characteristics. Nodes can mimic other protocols or they can behave randomly, but the traffic dispersion graph they generate must still reveal the overlay network they are organized in. Evidently, this structure will have to be correlated to malicious behaviour as well.

Accordingly, in the rest of the chapter we will focus on overlay traffic.

7.2 Network Monitoring with Traffic Dispersion Graphs

Approaches to automated traffic classification and filtering typically start from observing packets or flows and classify them through the application of a stack of methods ranging from simple port-based filtering to sophisticated supervised or unsupervised machine learning and classification methods over

packet and flow data. A summary of such methods is given in [93].

However, P2P botnet overlay traffic does not necessarily look malicious or harmful (in fact, in itself, it is neither), even if isolated and classified properly. It is essential to be able to identify this traffic as part of a *network* that, as such, makes it suspicious and could trigger a warning [23].

It has already been argued that it is very difficult to detect P2P traffic using packet or flow classification methods alone [50, 49]. Most of the key characteristics of P2P traffic lie in the network defined by the flows called the traffic dispersion graph (TDG). By building and analyzing TDGs of locally observable flow data after the classification phase, it is possible to extract important additional clues about the organization of an application and, for example, label them as P2P.

In [49], the TDG is defined on top of a set of flows S that have the usual format <srcIP, srcPort, dstIP, dstPort, protocol>. The TDG is the directed graph $G(V, E)$ where V, the set of vertices, contains the set of IPs in S, and E, the set of edges, contains the edges (a, b) such that there is a flow in S with srcIP $= a$ and dstIP $= b$.

Since our study intends to challenge the feasibility of local methods for detecting botnets, in order to be convincing we need to be generous to the local methods that are available for traffic classification. Therefore we assume that *traffic that belongs to a given P2P botnet can be isolated in an unlabelled way*. That is, we shall assume that there are methods available to group a set of flows together that belong to the P2P botnet, but that we cannot determine whether the identified class of traffic is in fact botnet traffic.

Note that this is already an extremely strong assumption. We will argue that even with this assumption, it is very difficult to identify the traffic as P2P traffic generated by a large P2P network if a P2P botnet applies certain P2P techniques. To show this, we will examine how the P2P traffic identification approach presented in [50] performs in this context.

7.3 Our P2P Overlay Model

As we saw in the Section 5.1, there are a huge number of design options, so selecting a suitable model that allows us to draw conclusions on the detectability of P2P botnet overlay traffic is highly non-trivial. Here we provide a short evaluation of the alternative design options and, based on this, we propose a simple model. After, we will present techniques that could help a P2P botnet to hide; we will examine these techniques in experiments described in Section 8.3.

Superpeer Overlays Since superpeer networks are more visible, and less robust to targeted attacks, we will assume that the most efficient botnets are not likely to adopt this design.

Structured P2P Overlays Current P2P botnets are based on structured overlays such as Kademlia [32].

Unstructured P2P Overlays Unstructured networks are extremely robust and, due to their lack of structure, they can be harder to discover as well. However, command and control operations are more expensive; and, most importantly, communication cannot be localized (an important technique will be described below) since we have no structure to map on the underlay. Although we do not rule out unstructured networks as a potential architecture for botnets, here we focus on structured networks.

7.3.1 Our Model

As a model we shall use an ordered ring with exponential long-range links, which is a simplified version of the Chord topology [119]: We have N nodes with IDs $0, 1, \ldots, N-1$. Node i is connected to nodes $i-1 \pmod N$ and $i+1 \pmod N$ to form the ring. In addition, node i is connected to nodes $i+2^j \pmod N$ for $j = 1, 2, \ldots, (\log_2 N) - 1$, which are the long-range links.

It is important to make a distinction between the overlay and the flows that exist in the overlay. Two nodes a and b are connected in the overlay if a "knows about" b. This, however, does not imply that a will ever actually send a message to b. For example, a might remember b simply in order to increase robustness in the case of a failure. On the other hand, a node a might send a message to b even though b is not the neighbour of a in the overlay (that is, for the overlay to function properly, a does not need to remember b after sending it a message). For example, in Kademlia if a wants to find the node of ID x then a will actually make contact with all nodes on the route to x. This is why Storm bots generate so many messages locally as part of the overlay traffic [40].

In short, we want to model the flows and not the overlay *per se*, so our model refers to the *flows* we can potentially observe. In the actual overlay there would probably be links to the 2nd, 3rd, etc, neighbours in the ring as well as those are learned from direct neighbours.

In the following we will describe two fairly straightforward techniques that future botnets could use to hide their traffic. The key point is that, using these techniques, the functionality of the overlay can be preserved while using far fewer links and traversing fewer routers.

Clusters for Sharing Long Range Links

In the ring every node has two neighbours that it actually communicates with at any given time, but it has $\log N$ long-range links, all of which are frequently used for communication to achieve as few as $O(\log N)$ hops in overlay routing (where N is the network size). Evidently, the ring would be

sufficient for communication but then sending a message from a node to another random node would require $O(N)$ hops in expectation.

There is a middle ground: we can reduce the number of long-range links to a constant number, and still have relatively efficient routing: $O(\log^2 N)$ hops [85] or even $O(\log N)$ hops [84].

However, let us remember that we are interested in the flows and not the overlay. In fact, we can modify our model to have a *single* long-range flow per node, and still have $O(\log^2 N)$ hops for routing messages in expectation. The trick is to create clusters of $\log N$ consecutive nodes in the ring, and allow each node to actually use only one of its long-range links. Routing proceeds as usual: but when a node decides to send a message over a long-range link, it first has to locate the node in its cluster that is allowed to use that link and send the message to that node along the ring. Note that nodes that are in the same cluster can rely on an identical set of long-range links since clusters can be interpreted as replicas of a node in an overlay of size $N/\log N$.

Next, we state without proof that a much simpler stochastic approach in which we have no clustering at all, but where each node can use only one random long-range link results in a similar routing complexity in expectation. Here a node has to look at its $\log N$ neighbours in the ring and pick the best long-range link that is allowed in some of these neighbours.

In sum, from the point of view of flows all nodes now have two ring flows (one in and one out) and two long-range flows on average (one out and one in on average).

Locality

We can also optimize the ring by trying to assign IDs to nodes in such a way that the resulting ring has links which touch the smallest possible number of routers. Several algorithms are known to achieve such optimized topologies that could be adapted to this application, such as that described in [88, 59].

7.4 Simulation Experiments

To examine the partial TDGs as seen locally from several points of the Internet, we (i) created a static AS-level model of the Internet topology and routing, (ii) mapped the overlay network to this AS-level topology, (iii) and we analyzed the local TDGs that are defined for each AS by the set of traversing flows in our model. We will now elaborate on these steps.

7.4.1 The AS-level Underlay

As our AS-level underlay we used an AS link dataset from CAIDA [47] that we cleaned by deleting uncertain links (around 3% of all links). We are aware of the methodological problems associated with collecting AS-level links and simulating protocols over them. However, for the purposes of this study, the main goal was not to achieve perfect low-level realism but to capture the important structural properties of the Internet as a complex network, a level that even a good topology generator could provide.

We calculated the shortest paths for each pair of nodes in the topology after assuming that the edges have equal weights. As a simple model of BGP routing we assumed that flows actually follow these shortest paths. Shortest paths also define the betweenness centrality of each node; that is, the number of shortest paths that touch a given node. This is a very important metric from the TDG point of view, since an AS with a high betweenness value is likely to be able to capture a more complete view of the TDG of the application.

The statistical properties of the AS graph have been studied intensively (see, for example, [83]). Figure 7.1 illustrates the distribution of betweenness centrality in the dataset we used, which is presented because we found the sharp switch to an exponential relationship at rank 1700 interesting.

7.4.2 Mapping the Overlay to the Underlay

Based on notions described in Section 7.3, we will experiment with two kinds of mappings: random and localized. First we describe the common settings for these two mappings, and then we will discuss the specifics of both.

Common Settings

In all our experiments the overlay contains 100,000 nodes. Note that we do not expect our results to be sensitive to the overlay size, since the overlay localization techniques we discussed in Section 7.3 essentially cause the problem to depend only on the AS-level graph.

The AS topology contains 14,630 nodes. With each type of overlay we map the overlay nodes onto the AS nodes in such a way that the number of overlay nodes in each AS is proportional to the size of the AS, but each AS has at least one overlay node. That is, in our model we do not take into account the geographical, social, or cultural bias that is known to affect botnet distribution [40]. The size of an AS is approximated and based on the IP-prefix-to-AS mapping available from CAIDA`http://www.caida.org/data/routing/routeviews-prefix2as.xml`.

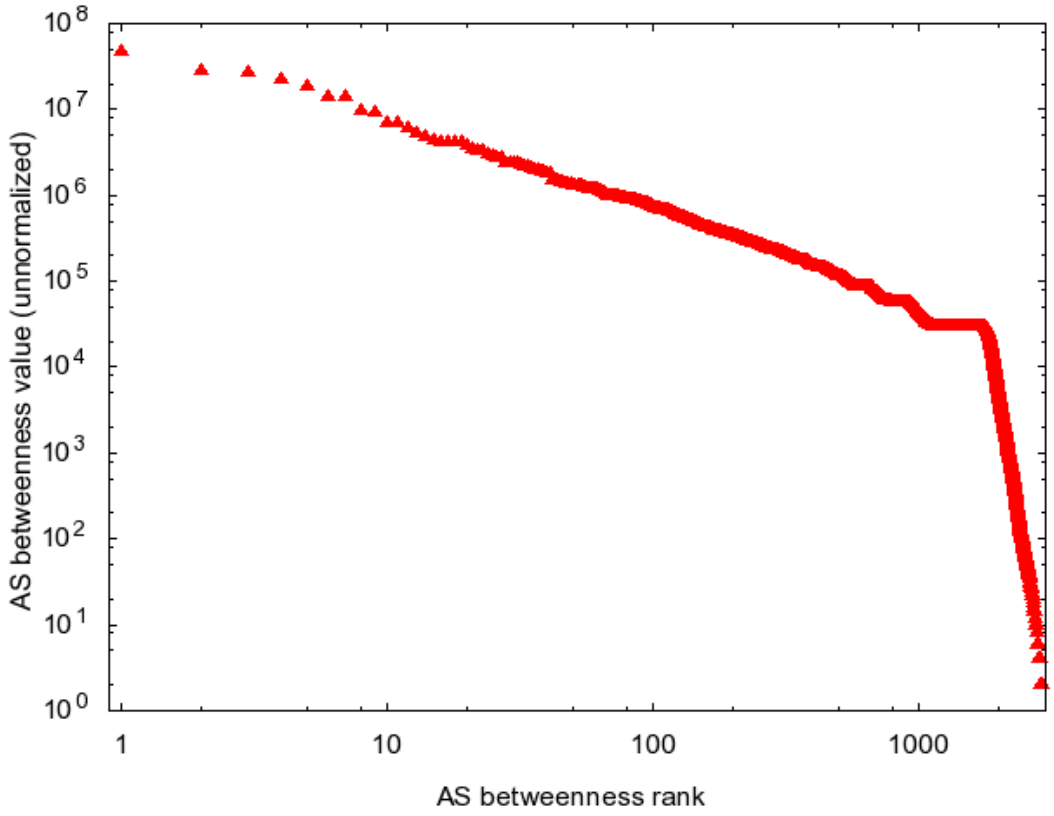


Figure 7.1: The power law relationship between betweenness rank and value in our dataset. After the sharp drop at around rank 1700, the relationship becomes exponential (not shown).

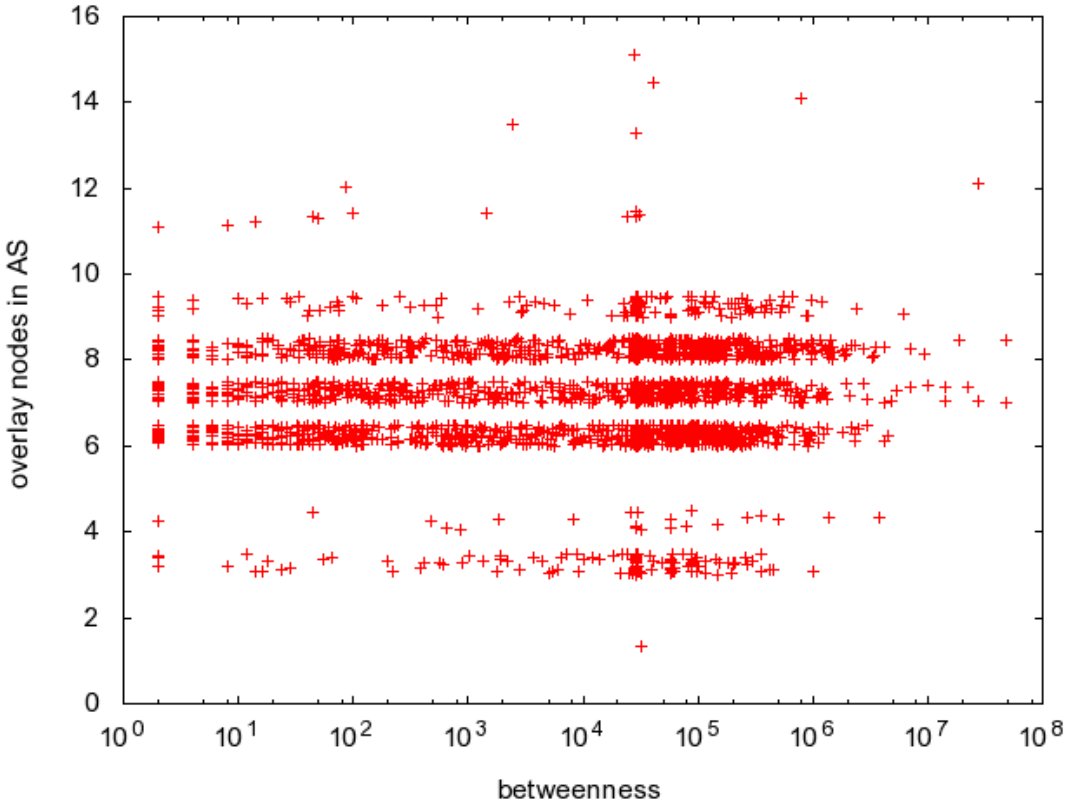


Figure 7.2: Scatter plot of the number of overlay nodes in an AS and the betweenness centrality of the same given AS.

It is interesting to note that the size and the betweenness of an AS seem to display no correlation, as illustrated by the scatter plot in Figure 7.2.

Specific Mappings

In the *random mapping* we assign overlay nodes to ASes at random, keeping only size proportionality in mind, as outlined above.

To create a *localized mapping* like that described in Section 7.3.1, we first define the travelling salesperson problem (TSP) over the AS topology and, using a simple heuristic algorithm, we produce a "good enough" tour over the ASes. Then, we assign the overlay nodes to ASes in such a way that the overlay ring is consistent with this tour, as illustrated in Figure 7.3.

We define the TSP as follows: find a permutation of the ASes such that if we visit all the ASes exactly once in the order given by the permutation, but assuming a closed tour that returns to the

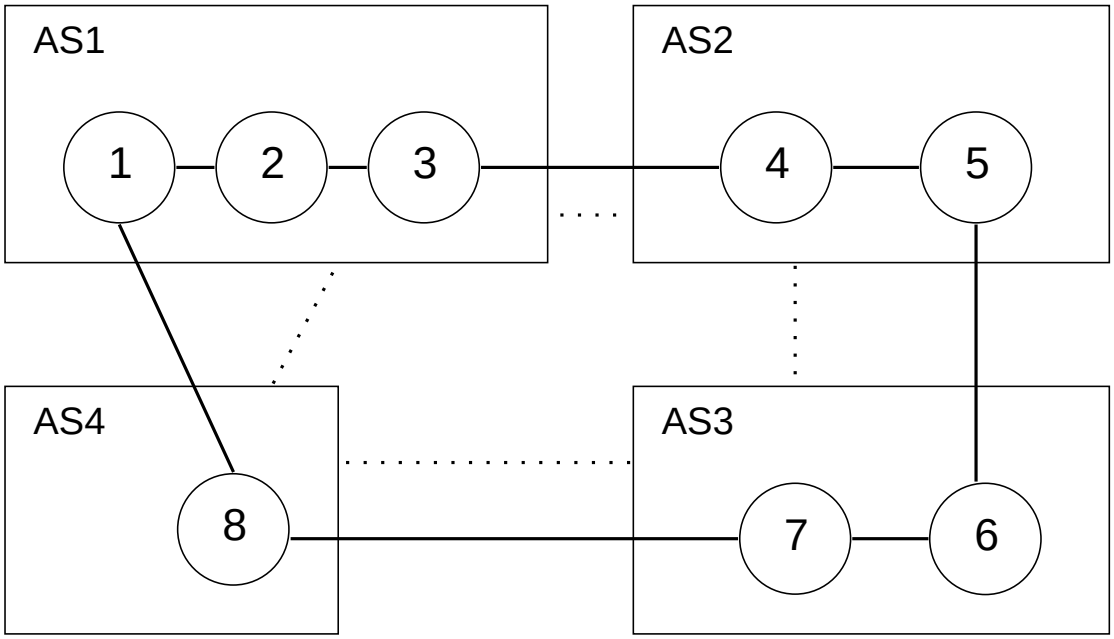


Figure 7.3: Localized mapping: alignment of the overlay ring (solid lines and circles) with the AS tour (dotted lines and rectangles) that is output by the nearest neighbour heuristic.

origin, and assuming also that for each transition from one AS to the other we follow the shortest paths in the AS-level topology, then the *sum of the hops* in the AS-level topology is minimal.

The heuristic we applied is nearest neighbour tour construction [63]. We start the tour with a random AS, and iteratively extend the tour by adding an AS that has the smallest shortest path length among those ASes that have not yet been visited. Ties are broken at random.

Before moving on to the analysis of TDGs, some comments are in order. First, our simulation is completely indifferent to the way a solution for the TSP problem is generated (i.e. a P2P algorithm or some other arbitrary heuristic method). What we would like to focus on is what happens when the mapping is well localised.

Second, the heuristic mapping we produce is most likely quite far away from the optimal localization. The actual optimal mapping is prohibitively expensive to calculate since the TSP problem is NP-hard in general, and we have a very large instance. Moreover, the definition of the localization problem itself could be refined as well, taking the requirements of the P2P botnet into account more directly in the objective function, and, for example, minimizing the sum or the maximal number of flows that can be seen at the ASes.

For these reasons our results should be interpreted as an upper bound on the amount of infor-

mation that is available at local nodes.

7.4.3 Analysis of TDGs

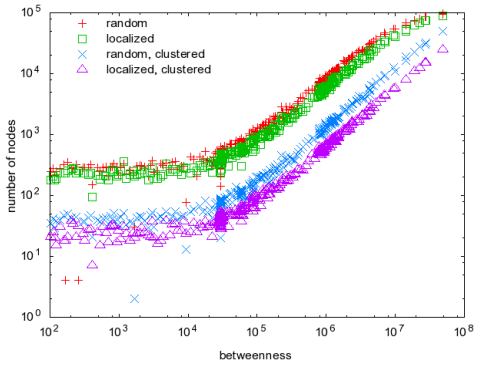
We experimented with four overlay models that are given by the two kinds of mappings described in Section 7.4.2 (random and localised) with or without the clustering technique described in Section 7.3.1. For these models we simply collected the flows that traverse a given AS, created the TDG, and collected statistics. The statistics we collected were the following: number of nodes, number of edges, number of weakly connected components, size of the largest weakly connected component, average node degree (where we count both incoming and outgoing connections) and finally, a metric called InO, introduced in [50]. InO is the proportion of nodes that have both incoming and outgoing connections. The results are shown in Figure 7.4. The first observation we can make is that the more efficient factor for hiding the overlay traffic is clustering. Recall that the main effect of clustering is to reduce the flows each node participates in from $O(\log N)$ to 4 on average. The effect of localisation is significant as well, but it is less dramatic overall. There is one exception: the largest connected component, where localization results in a value that is two orders of magnitude smaller than that for the two most central ASes.

Let us first compare these results to those found in [50] for existing P2P networks in real traces. There it was concluded that P2P traffic can be characterised by a high InO value (greater than 1%) and a high average degree (greater than 2.8). From this point of view, the TDGs we observe *can not be classified as P2P traffic* because the average degree is extremely low: in fact fewer than 2 in the case of a localized and clustered network even for the most central ASes. It is interesting that even for random mapping without clustering the threshold is crossed only at the most central ASes, although by a large margin.

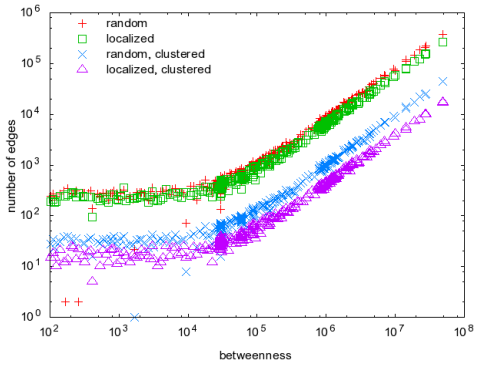
On the other hand, the InO values are large. This is simply because we did not concentrate on calculating this metric explicitly. The reason is that in practice determining the direction of a flow is not very reliable, it is prone to errors and is quite easy to manipulate. We predict that the InO value could also be manipulated by a botnet using techniques that cannot be captured by the relatively high level model we apply that ignores flow details and dynamics.

In addition, in [50] some applications with high InO and low average degree have been found: one example is FTP, where the server initiates connections to the client as well, which further complicates detection and offers the botnet other opportunities for camouflage.

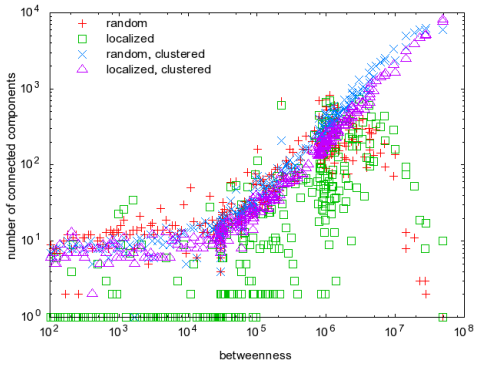
Of course it is possible that other metrics could help characterize these TDGs for a P2P network. Let us look at the TDGs using other metrics in order to get a more precise idea of what information is visible locally. Out of the 200,000 edges in the overlay, even the most central AS can see only



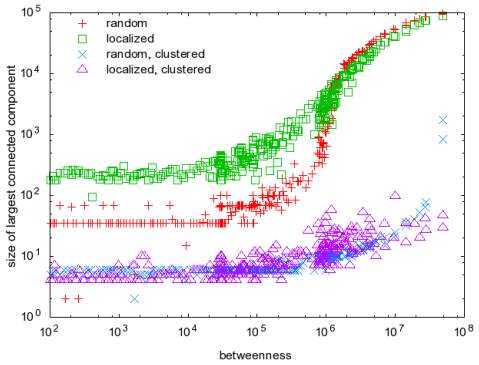
(a) Number of nodes



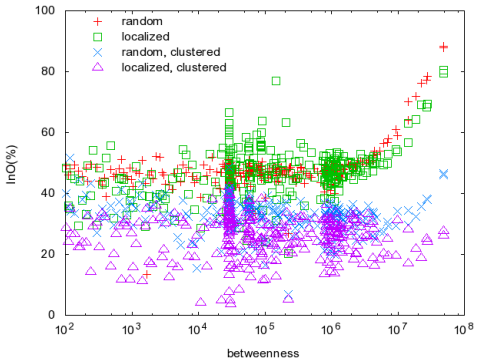
(b) Number of edges



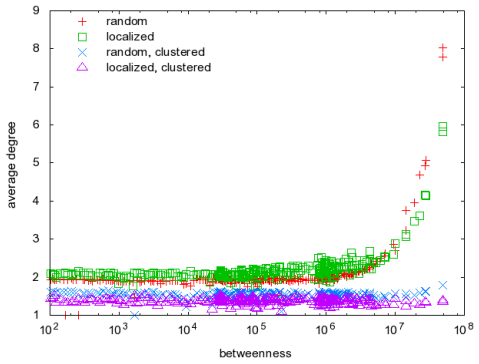
(c) Number of clusters



(d) Size of the largest cluster



(e) InO



(f) Average degree

Figure 7.4: Different characteristics of the TDG as seen from different ASes of a given betweenness.

16,814 edges at most. The number of nodes in the TDG is 24,985, which is much larger than the number of edges: indeed, the connected components are mostly of size 2 (pairs) and 3. There are 8,172 clusters, the maximal of which contains only 29 nodes. A visualization of the TDG that belongs to the most central AS is shown in Figure 7.5. The information available at the less central ASes is significantly less, as shown in figures 7.4 and 7.5. Lastly, the maximal node degree we observed in any TDG we have generated is no more than 4.

It is important to emphasize that results presented here are based on the assumption that within one AS transit, traffic traces can be aggregated and treated in a unified way. Although not impossible, this is a rather strong assumption, especially for the most interesting ASes with high betweenness centrality, which handle enormous volumes of transit traffic. In practice, the information visible locally could be even more fragmented.

Overall, then, we may conclude that when localization and clustering are applied, the overlay network traffic is almost completely hidden. A non-trivial proportion of the traffic can be seen only at the most central ASes, but even there, what is visible is predominantly unstructured.

7.5 Conclusions

Instead of looking at existing P2P botnets, we created synthetic flow data to model the set of flows that are available at an AS locally for observation. While it is clear that this methodology involves a simplified model of communication, it does allow us to get ahead of botnets by experimenting with algorithms that have not yet been deployed.

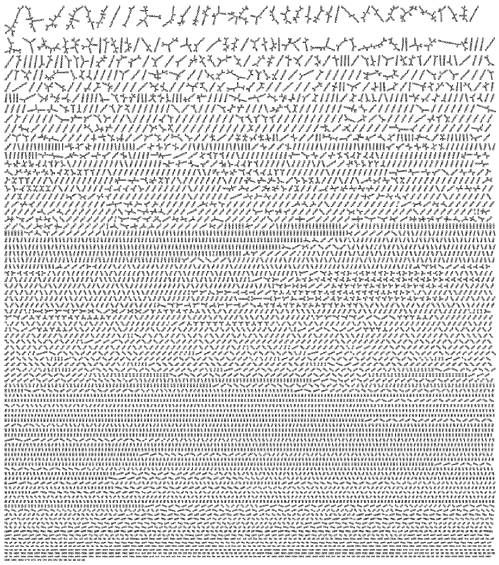
In spite of the low resolution that this methodology offers, we were able to predict and analyze a real problem: P2P overlays that are capable of efficiently and robustly organizing and controlling a large set of bots with a minimal communication footprint so as to avoid automated detection.

We hope that our results will provide non-trivial clues concerning directions for future research in automated botnet detection. Our results also show that we need to fight fire with fire and develop and apply P2P technology over large sets of cooperating administrative domains. In this chapter we presented an example for infrastructure awareness. Botnets can hide with the help of appliying a low degree P2P overlay. The question is how one can construct a low degree, but robust P2P overlay. We will study this in the next chapter.

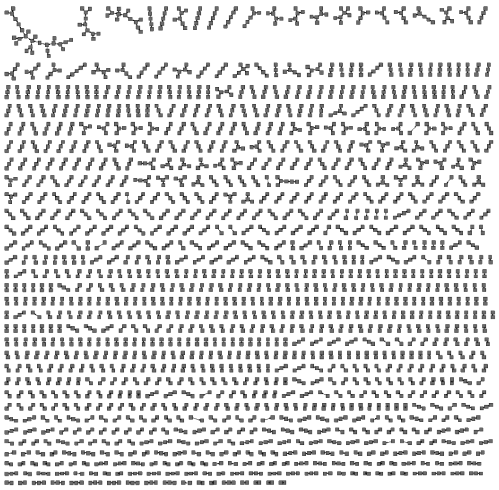
The results of this contribution point were published in research paper [JB09b].

Theses:

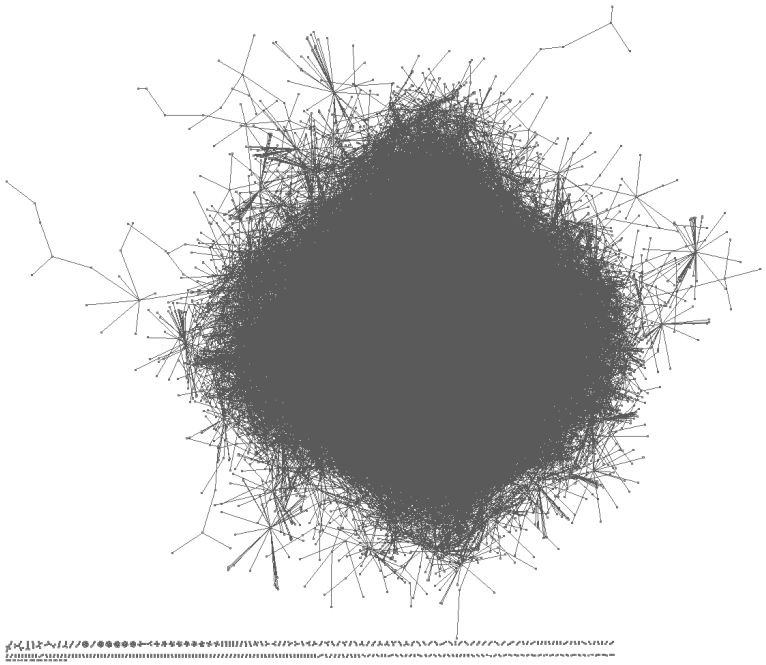
Thesis 3 *It is possible to build P2P botnet which cannot be detected with the help of the TDG*



(a) AS174 (betweenness 48,904,554), localized, clustered



(b) AS3491 (betweenness 4,460,142), localized, clustered



(c) AS3491 (betweenness 4,460,142), random

Figure 7.5: Visualizations of TDGs at various ASes. AS174 had maximal betweenness in the dataset.

method applied from a single point (AS).

Thesis 4 *Localization has only a minor effect on the hiding, the link clustering has a significant effect on the visibility of a P2P botnet.*

Thesis 5 *With the help of localization and link clustering a P2P botnet can hide from a single point TDG-based monitoring.*

The results connected to the Thesis 5 are the results of the author. The rest are the results of shared work.

8

Small degree DHT with large churn

In the last chapter we showed that state-of-the-art techniques for the detection of P2P networks fail if peers communicate with only a small constant number of neighbours during their lifetime[JB09b].

Fortunately, current infections generate considerable traffic. For example, the Storm worm touches a huge number of peers, in the range of thousands [32], when joining the network, generating a recognizable communication pattern as well as revealing a large list of botnet members. In general, P2P clients are typically in contact with a large number of neighbours due to maintenance traffic, and regular application traffic such as in a search. There are only a few notable exceptions, such as Symphony and Viceroy [85, 84], which are overlay networks of a constant degree.

We also saw in the chapters 4 and 3 that the number of flows has an important impact on the stateful services provided by the active devices. Therefore the number of flows initiated and maintained by a P2P node is important from the point of view of ISP friendliness as well.

It is still an open question of whether it is possible to create overlay networks of a very small constant maximal degree that are both efficient and scalable. Research activity concerning Symphony or Viceroy has not yet been targeted to the lower end of maximal node degree, potentially as small as 3 or 4. In fact, even negative results are known that indicate the inherent lack of scalability of constant degree networks [75, 66]. In this chapter we will answer this question in the affirmative and show that it is possible to build a Symphony-inspired overlay network of a very small constant degree, and with the application of a number of simple techniques, this overlay network can be made scalable and robust too. This result suggests that more research should be done into the detection

of malicious P2P networks that are potentially of a small maximal degree.

Our contribution is threefold. First, we empirically analyze known as well as new techniques from the point of view of improving the search performance in a Symphony-like overlay network. Second, we present theoretical results indicating that if we add $O(\log N)$ (or, in a certain parameter range, $O(\log \log N)$) backup links for all links (where N is the network size), then a constant degree network becomes fault tolerant even in the limit of infinite network size, while its effective degree remains constant (that is, the network can remain in stealth mode) since the backup links are not used for communication unless they become regular links replacing a failed link. This result is counter-intuitive because routing in Symphony requires $O(\log^2 N)$ hops on average. Third, we provide event-based simulation results over dynamic and realistic scenarios with a proof-of-principle implementation of a constant degree network, complete with gossip-based protocols for joining and maintenance.

8.1 Performance of Small Constant Degree Topologies

Our motivation is to understand whether a reasonable routing performance can be achieved in a network that operates in stealth mode; that is, where the maximal node degree is as small as 3 or 4. We will base our discussion on Symphony, a simple, constant degree network [85]. We explore several (existing and novel) simple techniques for improving the routing performance of Symphony at the lower extremes of maximal degree. To the best of our knowledge, this problem has not been tackled so far in detail by the research community.

Symphony was proposed by Manku et al. [85] as an application of the work of Kleinberg [73]. Like many other topologies, the Symphony topology is based on an undirected ring that is ordered according to node IDs. Node IDs are drawn uniformly at random from the interval $[0, 1]$ when joining the network. Apart from the two links that belong to the ring, each node draws a constant number of IDs with a probability proportional to $1/d$, where d is the distance from the node's own ID. After, each node creates undirected long-range links to those peers that have the closest IDs to the IDs drawn. (We note that an implementation needs an approximation of the network size N to normalize the distribution. A rough, but practically acceptable, approximation exploits the fact that the expected distance of the closest neighbours in the ring is $1/N$.)

Symphony applies a greedy routing algorithm: at each hop the link is chosen that has the numerically closest ID to the target. Due to the undirected ring, the procedure is guaranteed to converge. It can also be proven that routing takes $O(\log^2 N)$ hops on average; the idea of the proof is to show that it takes $O(\log N)$ hops to halve the distance to the target.

In the following we describe techniques for reducing the number of routing hops in small constant degree networks. After, we will systematically analyze these techniques via simulations.

Lookahead. Greedy routing can be augmented by a lookahead procedure where nodes store the addresses of the neighbours of their neighbours locally as well, up to a certain distance. This way, route selection is based on the best 2, 3, etc., hop route planned locally as opposed to a 1-hop route. Routing with a single hop lookahead has been studied in detail [86, 91]. Since small constant degree networks have small local neighbourhoods that can easily be stored and updated, we will study 2-hop lookahead as well.

Degree balancing. To enforce a strict small upper bound on node degree, nodes that are already of maximal degree have to reject new incoming long-range links. To make sure that most joining nodes can create long-range links, we need to introduce balancing techniques. In addition to the usual technique of repeated join attempts, we propose degree balancing: when a node of maximal degree receives a join request, it first checks its closest neighbours in the direction of increasing node ID to see whether they have free slots for a link. This need not require extensive communication as neighbour information is available locally (and, for example, the lookahead mechanism described above also requires local neighbourhood information).

Stratification. Since each node has only a small constant number of long-range links (1 or 2 in our case), many hops will follow the ring. It is therefore important that neighbouring nodes in the ring have different long-range links. We propose a stratified sampling technique that involves dividing the long-range links into a logarithmic number of intervals $[e^i/N, e^{i+1}/N]$ $(i = 0, \ldots, [\ln N] - 1)$. All the nodes first choose an interval at random that is not occupied by a long-range link at a neighbour, and then they draw a random ID from that interval with a probability proportional to $1/d$, where d is the distance from the node's own ID.

Short-link avoidance. Interestingly, if the average route is long, then it might be beneficial to exclude long-range links that are "too short". This way we introduce some extra hops at the end of the route, when routing follows the ring only. However, we save hops during the first phases due to the longer long-range links. As we will see later, this technique works well only in very small degree networks where routes are long, but in such cases we can obtain a significant improvement. We implement short-link avoidance based on the same intervals defined for stratification above. We will introduce a parameter m, the number of shortest intervals that should be excluded when selecting long-range links. For example, for $m = 2$, the first possible interval will be $[e^2/N, e^3/N]$.

Network size	2^i, $i = 10, 11, \ldots, 20$
Maximal degree (k)	3 or 4 (1 or 2 long-range links)
Lookahead	0, 1, or 2 hops
Stratification	yes or no
Join attempts	1, 2, or 4 attempts
Degree balancing	1, 5, 10, or 20 neighbors checked
Short-link avoidance (m)	0, 1, 2, 3, or 4

Table 8.1: The parameter space of the experiments.

We performed experiments using the parameter space defined in Table 8.1. For each parameter combination, we first constructed the network and after we selected 10,000 random node pairs and recorded the hop count of the routing.

The main methodological tool we apply to evaluate the large parameter space is drawing scatter plots to illustrate the *improvement* in the hop count as a function of a varying parameter. In these plots the points correspond to different combinations of the possible values of a subset of parameters. The remaining free parameters are the ones we are interested in; they are used to calculate the coordinates of the points as follows. The hop count for a specified setting for the free parameters is the horizontal coordinate of a point, whereas the vertical coordinate is the ratio of the horizontal coordinate and the hop count that belongs to another (typically baseline) setting of the same parameters.

The improvement brought about by stratification is illustrated in Figure 8.1 (left). Clearly, for almost all parameter settings, stratification is a better choice (most values fall below 1). The experiments with values higher than 1 were performed on the smallest networks, with no apparent additional common features. The lack of improvement in these cases is most likely due to the larger noise of random sampling in smaller networks. From now on, we will restrict our discussion to experiments with stratification.

The improvement brought about by lookahead is shown in Figure 8.1 (right). We can see that lookahead helps more if the degree of the network is larger. This is plausible since the local neighbourhood is exponentially larger in a network of larger degree. We also notice that lookahead is more useful in larger networks where the routes are longer.

Let us now take a look at the average degree of the networks (Figure 8.2). The main observation here is that it is important to approximate the maximal degree because in some cases we can observe a performance improvement of almost 30% relative to the baseline approach (that is, when no balancing efforts have been made), especially if lookahead has been applied as well. The setting of 2 join attempts with degree balancing over 10 neighbours appears to be a good compromise between

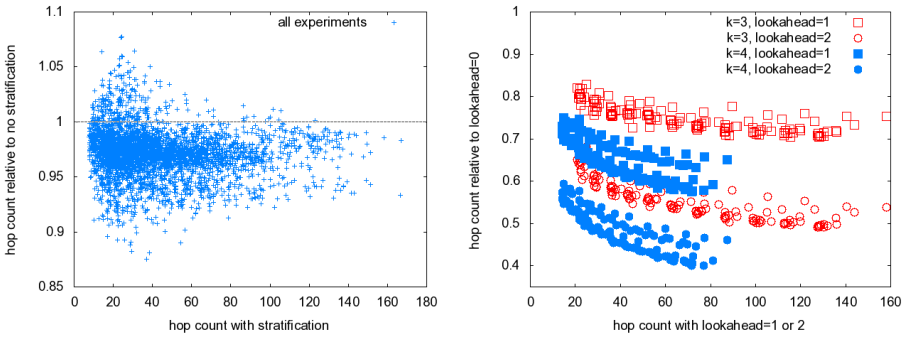


Figure 8.1: The improvement in hop count as a result of stratification (left) and lookahead (right).

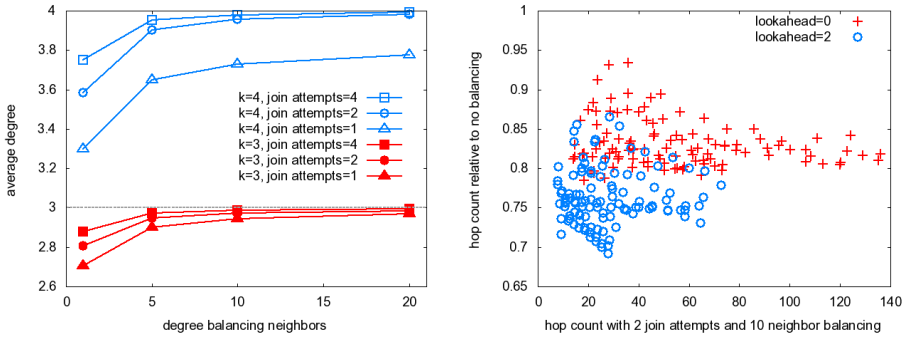


Figure 8.2: Average degree for $N = 2^{20}$ (left) and the performance improvement achieved by a good balancing strategy (right). The average degree is practically identical with all the other network sizes too (not shown).

cost and performance.

Figure 8.3 illustrates the effects of short-link avoidance. When $m = 2$ (left), the performance is improved with each parameter setting, except for $k = 4$ and $lookahead = 2$, where routing is so efficient that even for the largest networks there are too few hops, so short-link avoidance does not result in a net gain in hop count. For $m = 3$ (right) the same effect is amplified: for parameter settings with a large hop count the relative improvement is larger, but for short routes the relative cost is larger as well. All in all, the effect of this technique depends on the other parameters, but $m = 2$ appears to be rather robust and results in a slight improvement in most settings. We note that $m = 4$ was not the best setting in any of the experiments, so the maximal reasonable value was $m = 3$ in our parameter space.

Lastly, Figure 8.4 shows the hop count as a function of network size. Theory predicts an $O(\log^2 N)$ hop count complexity; to a good approximation this scaling behaviour is apparent, especially for $k = 4$.

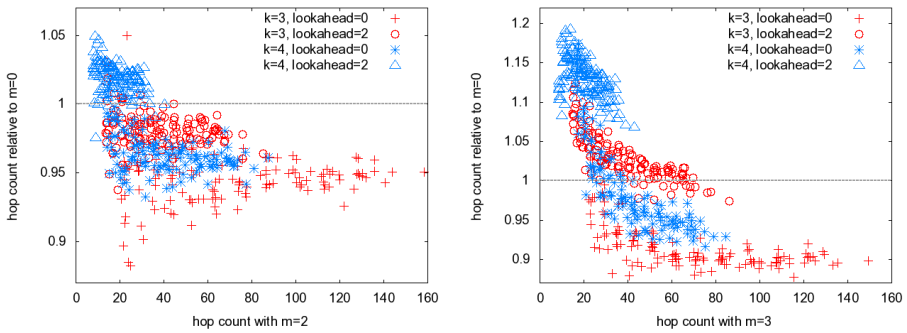


Figure 8.3: The improvement in hop count as a result of setting $m \neq 0$.

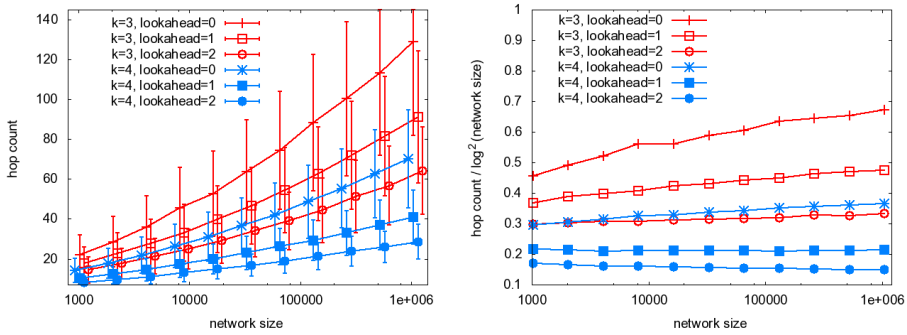


Figure 8.4: Scalability of routing. Statistics over 10,000 random node pairs are shown. Stratified sampling was applied, along with 2 join attempts with a degree balancing over 10 neighbours, and we set $m = 2$.

The very large difference between the best and the worst setting is also worth noting. Moving from $k = 3$ to $k = 4$ results in a very significant improvement: 2 long-range links instead of 1 causes the speed of routing to double, as predicted by theory [85].

We may conclude that routing in networks of a very small constant degree is feasible if certain techniques are applied. We found that the most effective technique is lookahead based on locally available information in the neighbourhood of the nodes. In addition, degree balancing is very important as well. Further techniques such as short-link avoidance and stratification also result in an additional 5-10% improvement, depending on the parameters. With these techniques we can route in around 30 hops in a network of size $N = 2^{20} \approx 1,000,000$ with a maximal node degree of only 4.

8.2 Scalability of Fault Tolerance

In the previous few sections we discussed several aspects of scalability. Now we shall examine whether the *fault tolerance* of the network diminishes as the network grows. This is crucial from the point of view of P2P networks (in particular, botnets), which have to tolerate node churn as well as other types of failures.

We first touch on some important issues regarding the scalability of constant degree topologies, and then we propose the simple technique of using backup links to increase their fault tolerance. We present theoretical results to show that the proposed technique indeed makes constant degree networks scalable in a well-defined sense in the presence of node failures.

We consider properties of networks of size N as $N \to \infty$. This means that the results presented here are mainly of theoretical interest, since in practice an upper bound on network size can easily be given, and the algorithm designer can set protocol parameters according to the upper bound even if the algorithm is not scalable in the present sense. Still, our results are somewhat counter intuitive, and as such increase our insight into the behaviour of constant degree networks.

The Achilles' heel of constant degree networks is fault tolerance and not performance. Performance is not a problem if the network is reliable. It is well known that a constant number of neighbours is sufficient to build a connected structure. Not only the trivial constant degree topologies such as the ring or a tree are connected, but also there exist random topologies of constant degree such as random k-out graphs. In such graphs each node is connected to k random, other nodes. It has been shown that for $k \geq 4$ a k-out graph is connected with high probability [19]. It is also well known that a constant degree is sufficient for an efficient routing algorithm. In the Symphony network, routing takes $O(\log^2 N)$ hops, while in Viceroy, the optimal hop-count is $O(\log N)$ [84, 85].

Unfortunately, constant degree networks do not tolerate node failure very well. We will examine the case when each node is removed with a fixed constant probability q (that is, the expected number of nodes remaining in the network is $(1-q)N$). For example, the 4-out random graph is no longer connected with high probability in this model. In fact, in order to get a connected random topology in spite of node failures, one needs to maintain $O(\log N)$ neighbours at all nodes [66]. Similarly, it has been shown by Kong et al. [75] that—in this failure model—DHT routing is not scalable in Symphony, while it is scalable on other topologies that are able to find more alternative routes by maintaining $O(\log N)$ links at all the nodes. In the following, we summarize the results of Kong et al. for completeness and extend them to show how to achieve an effectively constant degree, yet scalable, topology.

Kong et al. examined the *success probability* of routing $p(h, q)$, the probability that in a DHT a node h hops away from a starting node will be reached by the routing algorithm under a uniform

node failure probability q [75]. Their criterion for scalability is

$$\lim_{N\to\infty} p(h,q) = \lim_{h\to\infty} p(h,q) > 0, \quad 0 < q < 1-\epsilon, \tag{8.1}$$

where $\epsilon > 0$, and h is the average routing distance in the topology under study ($h = O(\log^2 N)$ for Symphony). This expresses the requirement that increasing network size should not increase sensitivity to failure without limit. Given this criterion, the proposed methodology consists of finding the exact formula or a lower bound for $p(h,q)$ for a topology of interest, and then calculating the limit to see whether it is positive. To calculate $p(h,q)$, one can create a Markov chain model of the routing process under failure, and determine the probability of reaching the failure state.

Kong et al. proved that Symphony is not scalable. They showed that for each step the probability of failure is a constant (C), so

$$\lim_{h\to\infty} p(h,q) = \lim_{h\to\infty} (1-C)^h = 0. \tag{8.2}$$

However, if we assume that there are *backup* links for each link in Symphony, then the situation changes dramatically. We shall not go into details here about how to collect the backup links; Section 8.3 discusses an actual algorithm. From our point of view here the important fact is that the backup links are such that if a link is not accessible, then the first backup is the best candidate to replace it. If the first backup is down as well, then the second backup is the best replacement, and so on.

Recall that the Symphony topology consists of a ring and a constant number of shortcuts. For the ring, the notion of backup should be clear. A shortcut link is defined by a randomly generated ID: we need to find the numerically closest node in the network to that ID. The first backup in that case is the second closest node in the network, and so on. This notion can be extended to all routing geometries as well.

The backup links do not increase the *effective* degree of an overlay node: a DHT can use the original links if they are available, even if some of the backups were closer to the target. In fact, backup links are *never used* for communication, not even during maintenance or any other function, except when they become regular links after replacing a failed regular link. In addition, as we will explain in Section 8.3, backup links can be collected and updated during regular DHT maintenance without any extra messages.

It seems clear that backup links can make Symphony scalable. But how many of them do we need? In the following we show that $O(\log N)$ backup links are sufficient, and in some circumstances even $O(\log\log N)$ links will do.

Lemma 8.2.1 *If in a DHT routing network all the links have $f(N)$ backup links then $p(h,q) \geq (1-q^{f(N)})^h$.*

Proof The probability of being able to use the best link in the original overlay is $1 - q$. After considering the backups this probability becomes $1 - q^{f(N)+1} > 1 - q^{f(N)}$. Now, if we follow only the optimal link in each step then the probability of success is not smaller than $(1 - q^{f(N)})^h$. Clearly, $p(h, q)$ is no less than this value since it accounts for methods for routing around failed links as well.

Lemma 8.2.2 $\lim_{N\to\infty}(1 - q^{\log N})^{\log^k N} > 0$ *if* $0 \leq q < 1 - \epsilon$ *and* $k \in$.

Proof For $k <= 0$ the lemma is trivial. For $k > 0$, based on to Theorem 1 in [75], we need to prove that

$$\lim_{N\to\infty} q^{\log N} \log^k N < \infty$$

and the lemma follows. The convergence of the above expression can be proved by applying the l'Hospital rule on $(\log^k N)/q^{-\log N}$ a suitable number of times.

Lemma 8.2.3 $\lim_{N\to\infty}(1 - q^{\log\log N})^{\log^k N} > 0$ *if* $0 \leq q < \min(e^{-k}, 1 - \epsilon)$.

Proof We again need to prove that

$$\lim_{N\to\infty} q^{\log\log N} \log^k N < \infty.$$

Substituting $x = \log N$ we get

$$q^{\log x} x^k = x^{\frac{1}{\log_q e}} x^k = x^{\frac{1}{\log_q e}+k}$$

This means that we need $\frac{1}{\log_q e} + k \leq 0$ for convergence. Elementary transformations complete the proof.

Theorem 8.2.4 *The Symphony topology is scalable; that is,* $\lim_{h\to\infty} p(h, q) > 0$*, if (i) all the links have* $O(\log N)$ *backup links, or if (ii) all the links have* $O(\log\log N)$ *backup links and* $q \leq e^{-2} \approx 0.135$.

Proof A straightforward application of the previous lemmas for Symphony where $k = 2$; that is, $h = O(\log^2 N)$.

To sum up, we have shown that Symphony-like topologies can be made scalable by adding only $O(\log N)$ backup links for all the links, and under moderate failure rates even $O(\log\log N)$ suffices. This is rather counter-intuitive given that routing still takes $O(\log^2 N)$ steps. It is also promising, because these results suggest that collecting good quality backup links can dramatically improve scalability at a low cost.

8.3 Experimental Results

In this section we present proof-of-principle experiments with a simple implementation of a small constant degree network in realistic churn scenarios. Our goal is not to present a complete, optimized implementation, but rather to show that it is indeed possible to achieve acceptable fault tolerance and performance in realistic environments.

We performed the experiments using the PeerSim event-based simulator [96]. In our system model nodes can send messages to each other based on a node address. Nodes have access to a local clock, but these clocks are not synchronized. Messages can be delayed and nodes can leave or join the system at any time. The statistical model of node churn is based on measurement data [122], as we describe later.

Our goal was to design a protocol to construct and maintain a Symphony topology in a fault tolerant way, with backup links (see Section 8.2). To this end, we applied T-Man, a generic protocol for constructing a wide range of overlay topologies [60]. Here we briefly outline the protocol and the specific details of the present implementation. The reader is asked to consult [60] for more details.

In our experiments each node has a single long-range link; that is, the maximal effective degree is 3. Each node has three local caches: long-range backups, ring backups and random samples. We set a maximal size of 80 for both the long-range and ring backup caches, and 100 for random samples. These values were chosen in an ad hoc way and were not optimized.

The caches contain node descriptors that include the ID and the address of a node. Each node periodically sends the contents of all its caches to all its neighbours. The period of this communication is called the *gossip cycle*, and was set to 1 minute in our experiments.

As described in Section 8.2, the two backup caches should ideally contain those nodes from the entire network whose IDs are closest to the node's own ID (for the ring neighbours), and the ID of the long-range link, respectively. When receiving a message containing node descriptors, a node updates its own local caches. It also updates the random sample cache, using a stratified sampling approach: the ID space is divided into 100 equal intervals, and each cache entry is selected from one of these intervals. If the random sample cache can be improved using any of the incoming node descriptors, the cache is updated.

In addition, if a node receives a message from a node it should not receive messages (for example, because the sender has inaccurate knowledge about the topology) from the node it sends its caches to the sender of the misdirected message as well, so that it can improve its backup caches.

The join procedure starts by generating the node's own ID at random, as well as the ID for the long-range link. The caches need to be initialized as well, using a set of known peers; we applied 50 fixed descriptors for the initialization. Once the caches contain at least one link, the gossip protocol

sketched above can start, and all the caches will fill and improve gradually.

When handling a routing request, a node applies greedy routing using the three links: the two ring links and the long-range link. However, before using the currently active ring links or long-range link, the node always checks the best candidate in the backup caches for availability (sending a ping message). Note that we do not check the best candidate for the message to be routed; we check the best candidate for the given link slot (ring or long-range). This way, it is guaranteed that the right links are used based on the state of the network at any given time.

The scenario we experimented with involves node churn. Applying appropriate models of churn is of crucial importance from a methodological point of view. Researchers have often applied an exponential distribution to model uptime distribution, which corresponds to a failure probability independent of uptime. Measurements of a wide range of P2P networks in [122] suggest that a Weibull distribution of uptime is more realistic, with a shape parameter around $k = 0.5$. In this case the failure rate decreases; that is, the more time a node spends online, the less likely it is to fail. This favours longer sessions, but the Weibull distribution is nevertheless not heavy-tailed.

We applied the Weibull distribution with $k = 0.5$ to model uptime, and scaled the distribution, so that around 30% of the nodes live longer than 30 minutes [122]. The downtime distribution was modelled by a uniform random distribution, with an average downtime of 2 minutes. This average is very short; however, longer downtimes result in a relative increase in the proportion of nodes in the network that have long session lengths. Paradoxically, if the downtime is long, then the network is almost completely stable in the time range we are interested in (around 30 minutes).

As noted in [122], the lengths of the online sessions of a node correlate: there are nodes that tend to be available and nodes that are not. We assigned to each node a fixed session length from the distribution above, which remained fixed during the experiment.

We applied a 1-minute gossip cycle. Each experiment lasted for 40 cycles. The network gradually grew to its final size during the first 10 cycles, when we added each node at a random time. During the remaining 30 cycles churn was applied. The network sizes we tested were $N = 2^i$, $i = 10, \ldots, 14$. Parameter m (which controls short-link avoidance) was set to $m = 0$ or $m = 3$. Other features such as lookahead, stratification, and degree balancing were implemented later.

Figure 8.5 illustrates the speed at which the ring topology was formed despite continuous churn. The improvement of the backup links (80 links per node) for the ring links is also illustrated. It can be seen that most nodes collect good quality backups, but some of them seem to have no useable backups at all; these nodes have a very short session time and spend very little time in the network.

One of our main goals was to show that the effective degree—the number of nodes an average node actually communicates with—can be kept low. Figure 8.6 shows that indeed this can be

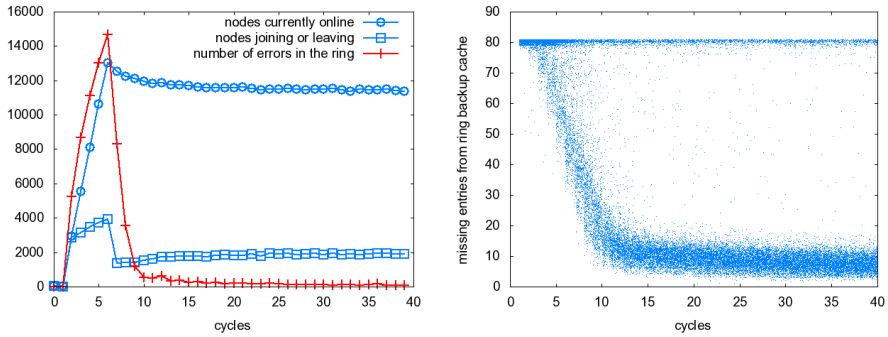


Figure 8.5: The evolution of the topology and the backup links for $N = 2^{14}$. In the figure on the right, points belong to individual nodes and have been randomly shifted so as to visualize the density.

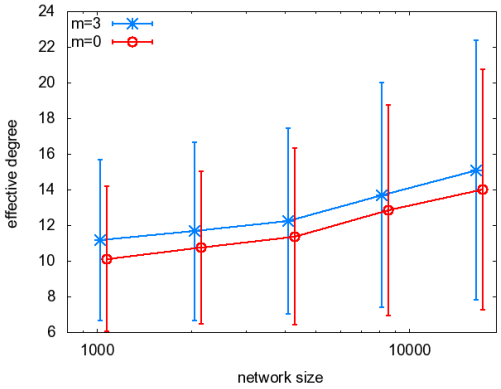


Figure 8.6: The observed effective degree by the end of the experiments.

accomplished. Despite heavy churn, which results in a constant fluctuation of the ring neighbours, the effective degree is small and seems to scale well. Recall that, for example, the Storm worm has been observed to communicate with thousands of neighbours [32].

Lastly, let us examine the reliability and the efficiency of routing (see Figure 8.7). Recall that we work with a baseline implementation with no lookahead, stratification, or any other technique. Only short-link avoidance has been implemented. When testing routing we pick IDs and not nodes as targets, and consider routing successful if the closest node receives the message at the time of reception. That is, it is possible that the optimal target is different at the start of the routing and at the end of the same routing.

We can see that short-link avoidance improves the hop count by a large margin. Overall, we observe almost twice the hop count as in the ideal case shown in Figure 8.4. However, in our hostile

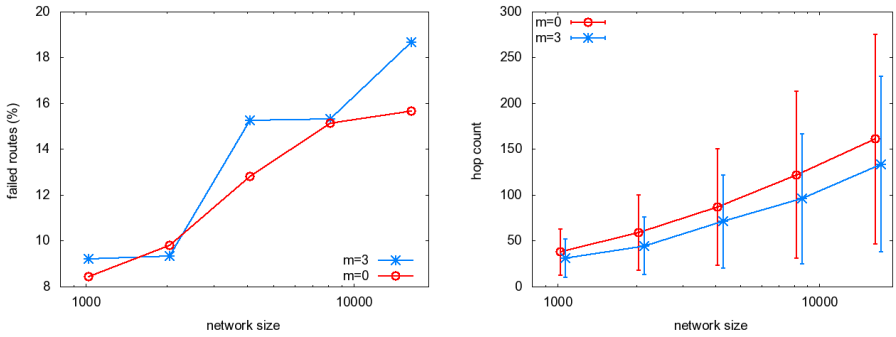


Figure 8.7: Routing performance. The figure on the right corresponds to successful routes.

scenario with heavy churn this can be considered acceptable for this baseline approach. We also note that the actual routing performance observed in real botnets can be significantly worse; for example, the success rate of queries has been found to be extremely low in the Storm botnet [32].

8.4 Conclusions

In this chapter we argued for the feasibility of P2P systems where nodes communicate only with a very limited number of peers during their lifetime.

Our results have at least two implications. First, they provide a strong indication that P2P botnets need to be taken seriously by the P2P community. In our previous work we showed that stealth mode P2P networks are practically invisible for state-of-the art methods for P2P network detection [JB09b]. Current botnets do not exploit P2P technology to its full potential, and by the time they learn how to do that, they will be very difficult to detect and remove.

The second implication is not related to malware. There can be other applications where it is important to utilize very few connections because of a large associated cost. We showed in the first part the cost of the large flow number. Further arguments for a constant degree design can be found in related works as well [84]. For this reason, the research issue that we raised; that is, the investigation of networks of a very small constant degree, is relevant to non-malicious applications as well. The results of this contribution point were published in research paper [JB09a].

Theses:

Thesis 6 *The Symphony topology is scalable; that is,* $\lim_{h\to\infty} p(h,q) > 0$*, if (i) all the links have* $O(\log N)$ *backup links, or if (ii) all the links have* $O(\log\log N)$ *backup links and* $q \leq e^{-2} \approx 0.135$.

Thesis 7 *It is possible to build a low degree DHT that is robust even in the case of large churn.*

Thesis 8 *The T-Man based DHT exetended with the four methods (Lookahead, Degree balancing, Stratification and Short-link avoidance) is stable in the case of significant churn (Weibull k=0.5)*

The results connected to Thesis 8 are the results of the author. The rest are the results of shared work.

9 SIP compression

9.1 Session Initiation Protocol (SIP)

We saw in Section 2.1.2 that the UMTS network is IP-based. Besides the similarities there are several differences between wired and mobile networks. The most important one is the bandwidth, which is the bottleneck of mobile core systems. The number and speed of processors, the memory, and the capacity of backing storages can be easily increased, but the communication speed among the units is limited.

Mobile system providers invested a huge amount of money into their systems. To get a good return on their investment, they have to provide acceptable services for customers. The customers would like to use these services independently of their location and hardware framework. There is a need for a communication protocol to check the available and required services and their parameters. Session Initiation Protocol (SIP) [35] was chosen by 3GPP for this purpose. It can be seen by now that SIP is one of the most important Internet protocols in the 3G mobile core system. However, it is highly redundant, because it is an extendable ASCII-based communication protocol. Recently, concerns has been raised in 3GPP that session setups in the all-IP network were too lengthy due to excessive signaling over the radio link. The delays were estimated to be as long as 10 seconds [36]. To overcome this problem, they agreed to use some kind of signaling compression along with UDP/IP header compression to shorten the transmitted messages. 3GPP raised the issue with IETF. After evaluating several proposals, the Robust Header Compressing Group (ROHC) [48] came up with the

idea of the universal decompressor, and defined a communication layer, called Signaling Compression (SigComp) for this functionality [100, 102, 101]. IETF has left many issues open or implementation specific, such as the negotiation of algorithms and parameters (e.g. buffer sizes). It is also not yet known how to integrate SigComp with the SIP protocol, which is the main goal in compression.

The SigComp layer consists of two main interworking entities: the compressor and the Universal Decompressor Virtual Machine (UDVM) [103]. There are several compression algorithms, and to allow one to freely choose any kind of them, SigComp is designed in such a way that it contains a universal decompressor. Thus not only the compressed messages can be sent, but also their compression algorithms, where necessary. UDVM has its own language for implementing and executing decompression algorithms; the only task is to upload the appropriate decompressing code to UDVM.

The protocol header compression is not a new idea. In 1990 Van Jacobson proposed a TCP/IP specific compression algorithm [58], sending between 3-5 bytes of the 40 byte header. Another interesting approach is suggested in [80]; it describes a universal framework which includes a simple platform-independent header description language that protocol implementers can use to describe high-level header properties, and a platform-specific code generation tool that produces kernel source code automatically from this header specification. In our case these approaches are unsuitable, because they are based on inter-packet redundancy. We have to deal with stand-alone packets because it is possible that we only have a one-way communication channel with an error prone link, so we do not know which packets arrived on the far side of communication channel. The classic compression algorithms are not applicable without modifications because of the short message lengths. We did not find any acceptable solution in the literature that met our special criterion. So we think that our study is the first to investigate the compressibility of the SIP protocol and examine in a general way the asymmetric[1] protocols needed to develop applicable compression methods.

In this chapter we will describe our work and the results concerning SIP compression. Our goal was to implement the SigComp layer and to study the SIP protocol so as to find the best compressing algorithm for it.

This chapter is organized as follows. First we will give an overview on SIP in Section 2, then we will briefly present the theoretical background of compression and some well-known algorithms. After, our work on compression algorithms will be described. Lastly, we will compare and evaluate our test results and state our main conclusions.

[1] Protocols without full handshaking

9.1.1 Brief description

There are many applications of the Internet that require the creation and management of a session, where a session is regarded as an exchange of data between an association of participants. The implementation of these applications is complicated by the practices of participants: users may move between endpoints, they may be addressable by multiple names, and they may communicate in several different media – sometimes simultaneously. Numerous protocols have been developed that carry various forms of real-time multimedia session data such as voice, video, and text messages. The Session Initiation Protocol (SIP) [31, 35] works with these protocols by enabling Internet endpoints (called user agents) to discover one another, and to agree on a characterization of a session they would like to share.

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls. SIP can also invite participants to existing sessions, such as multicast conferences. Media can be added to (and removed from) an existing session. SIP transparently supports name mapping and redirection services, which supports personal mobility and users can maintain a single externally visible identifier regardless of their network location. To locate prospective session participants, and for other functions, SIP allows the creation of an infrastructure of network hosts (called proxy servers) to which user agents can send registrations, invitations to sessions, and other requests. SIP works independently of underlying transport protocols and without any dependency on the type of session that is being established.

The changeover of IPv4 [99] and IPv6 [24] protocols has been studied and extensively analyzed, and turned out to be a necessary but not easy task [46, 114]. The 3G networks use SIP as their call control protocol and IPv6 as the network layer protocol for wireless communication, while currently IPv4 is used as the network protocol for the Internet. Thus a need has emerged to connect a mobile SIP user agent based on IPv6 with another SIP user agent based on IPv4. In a previous study [115] we created and tested a demonstration system, where the SIP communication between the IPv6 and the IPv4 networks was established between two SIP proxies (IPv6 and IPv4) via a special NAPT-PT. This study presented a new technique to ensure the communication between 3G mobile networks and Internet phones. As a continuation we would like to see how to integrate the SIP protocol – which is the main topic in compression – into the SigComp layer.

Instead of describing the whole functionality of SIP [20, 131], we shall present here a sample message, and a typical message flow.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
```

```
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com
   ;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com
   ;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com
   ;branch=z9hG4bK776asdhds ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
```

Figure 9.1 shows a typical example of a SIP message exchange between two users.

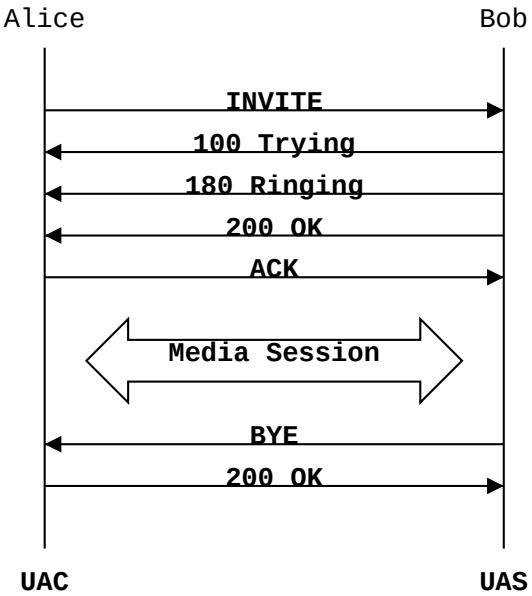


Figure 9.1: SIP session setup

9.2 Overview on Compression

9.2.1 Theoretical background

Data can be compressed whenever some symbols are more likely to occur than others. There are several theories for describing the information content of a data set. Entropy, as defined by Shannon, is the uncertainty regarding which symbols are chosen from a set of symbols with given apriori probabilities. If there is more disorder, or entropy, then more information is required to reconstruct the correct set of symbols [8]. Shannon's entropy is defined for a set of possible symbols S:

$$H = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}, \tag{9.1}$$

where p(s) is the probability of symbol s. If we have symbols $s \in S$, Shannon defined the notion of self-information of a symbol as:

$$i(s) = \log_2 \frac{1}{p(s)}. \tag{9.2}$$

This self-information represents the number of bits of information contained in it and, roughly speaking, the number of bits we should use to send that symbol. Entropy is simply a weighted average of the information of each symbol, and therefore the average number of bits of information in the set of symbols. This entropy (more precisely *First order Entropy*) gives us an upper limit for data compressibility when we do not know anything about partial transition probabilities between symbols, and we are coding only characters and only one message. Using Eq.(9.1), we have found that the entropy of a typical SIP message is around 6.7. The information can be coded with H bits/symbols (this can be achieved just with arithmetic coding). With this entropy we can calculate the achievable compression rate:

$$cr = \frac{compressed}{original} = \frac{6.7\ bit/char}{8\ bit/char} = 0.83. \tag{9.3}$$

How can we achieve a better compression ratio than Shannon's first order entropy? With second, third, ... nth order entropy. If we have some knowledge about the dependence between symbol probabilities and their context then we can calculate them with help of conditional probabilities. The conditional entropy for a set of symbols S and context set C is:

$$H(S|C) = \sum_{c \in C} p(c) \sum_{s \in S} p(s|c) \log_2 \frac{1}{p(s|c)} sip : 4 \tag{9.4}$$

When the conditional probability distribution of S depends on the context C, then $H(S|C) < H(S)$; otherwise $H(S|C) = H(S)$.

As we have seen, we can calculate upper limits for data compressibility if we know the symbol probability (conditional probability) distributions. But we would like to construct more effective compressing algorithms in practice. Hence we have to investigate the following aspects:

- Data modeling – If we know something about the data, we can provide an accurate probability model for each symbol (or group of symbols). This is mainly a dictionary, sorted by symbol likehood(or by the likehood of a set of symbols) . (The probability value in most cases is the relative frequency of a symbol in a message).

- Symbol coding – With help of symbol coding we can link the symbols (or group of symbols) to an approriate codeword (whose lengths depend on the symbol probability; the greater the probability, the shorter the assigned codeword).

We have to delimit the borders between codewords because they have different lengths. Symbol coding can be byte- or bit-based. In the case of byte-based coding, we can use special symbols; in the other case we can use prefix-free codewords to delimit the borders. A codeword is prefix-free if none of the codeword is a prefix of an another codeword.

9.2.2 Summary of some well-known algorithms

LZ77 is a byte-based run-length encoding algorithm. It scans the message and tries to find the largest equal parts with the dictionary. If it finds one, then it replaces it with the following pair of numbers: the starting address of the hit and the length of replaced character group. (Sometimes the distance of the hit from the beginning and the length is used.)

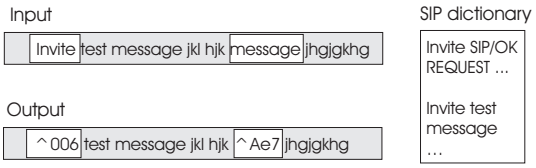


Figure 9.2: Illustration of how the LZ77 algorithm works

The following four algorithms are based on prefix-free encoding:

The Golomb-Rice encoder [44] is a prefix-free coding that assigns codes to the numbers 0,1,2,3,... according to the following description: first a power of 2 is chosen, denoted by 2^k. If we want to encode the number n, first $\lfloor n/2^k \rfloor$ 1 digits are written, a 0 after them and last a number $(n \bmod 2^k)$ that is k-length long even if it can be written with few digits, which corresponds to the following formula:

$$\text{code} = \frac{n}{2^k} \text{ unary code} + 0 + n \bmod 2^k \tag{9.5}$$

SubExponential This encoder [43] is also prefix-free and assigns codes to numbers. The main difference compared to the previous Rice encoder is that the lengths of codes grow logarithmically, depending on the numbers to be encoded.
First a power of 2 is chosen, denoted by 2^k. Next let n be the number we want to code. Then the code of n is:

- If $n < 2^k$, then the code of n is a 0 and n in exactly k-bit long.
- If $n \geq 2^k$, then the code of n starts $\lfloor \log(n) \rfloor - k + 1$ bits 1,
  a 0 and last the lowest $\lfloor \log(n) \rfloor$ bits of n.

The corresponding formula is:

$$\text{code} = \begin{cases} 0 + k \text{ unary code} & if n < 2^k; \\ \log(n - k + 1) \text{ unary code} + 0 + n \bmod \log n & if n > 2^k \end{cases} \tag{9.6}$$

Synth With help of Arithmetic compression [42] we can approach the best compression ratio using the Shannon entropy concept. The main idea behind arithmetic coding is to represent each possible sequence of n messages by a separate interval on the real line between 0 and 1. For a sequence of symbols with probabilities $p_1, p_2, \ldots, p_n$ the algorithm will assign the sequence interval size $\prod_{i=1}^{n} p(i)$ starting with an interval of size 1. We simplified this algorithm with fixed size intervals (fixed probabilities). The size of any interval is a power of 2.
The number n thus can be coded by the following rules:

- If $0 \leq n \leq 15$, then the code of n is a 0 (only 1 bit) and afterwards n is represented in 4 bits.
- If $16 \leq n \leq 31$, then the code of n is 10 (2 bits) and $(n - 16)$ is represented in 4 bits.
- If $32 \leq n \leq 63$, then the code of n is 110 (3 bits) and $(n - 32)$ is represented in 5 bits.
- If $64 \leq n \leq 127$, then the code of n is 1110 (4 bits) and $(n - 64)$ is represented in 6 bits.
- If $128 \leq n \leq 255$, then the code of n is 11110 (5 bits) and $(n - 128)$ is represented in 7 bits.
- If $256 \leq n \leq 767$, then the code of n is 11111 (5 bits) and $(n - 256)$ is represented in 9 bits.

Huffman encoding is an optimal prefix-free coding. The algorithm is based on the well-known Huffman tree. The Huffman-tree is built on the frequency of the keywords in the message, where each leaf contains a keyword. The messages are then compressed using this tree.

Deflate This algorithm [25] is a combination of two different compression algorithms. First the message is compressed by the LZ77 algorithm (or some variant) and then the encoded message is compressed by the Huffman algorithm.

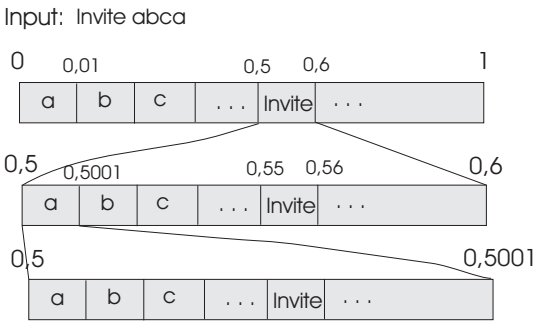


Figure 9.3: Illustration how arithmetic compression works

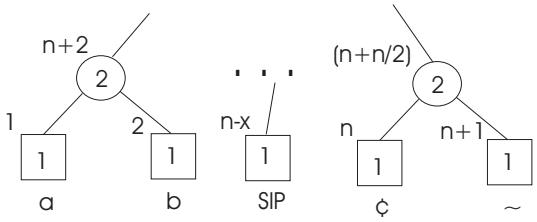


Figure 9.4: A sample Huffman tree

9.3 Our results

9.3.1 SIP compressibility

We present three approaches that are based on each other and indicate the level of SIP specificity.

- In our first attempts we treated the messages as if they were simple text-based messages, which was why we could not write efficient compression algorithms. A summary of our experimental results is given in Section 9.3.2.

- Afterwards we used dictionary-based algorithms, because we had some preliminary information about the message that can be incorporated into the compression algorithms as probabilities. We found a dictionary with SIP instructions ordered in terms of decreasing probability values [31]. We know that numbers and normal characters are more probable than special characters, and groups of symbols may appear more than once (run-length encoding). Some details about our dictionary are given in Section 9.3.3.

  We constructed several compression algorithms (mainly differing in symbol coding), starting from the well-known algorithms outlined in Section 9.2.2. These algorithms were modified so that they could make efficient use our dictionary. Moreover, we designed them so that they could be adapted. We used a kind of *Move-to-Front* algorithm to change the symbol

probability values. When a symbol occurred, we put it higher up the list in the dictionary. We used the dictionary both at the encoder and decoder stages, improving the efficiency of the compression. The messages could be compressed efficiently with this solution; sections 9.3.3 - 9.3.6 describe what we developed specifically for the various algorithms.

- The entropy of a message flow is better than the entropy of a message, so we can achieve better compression ratios when compressing message flow. Further improving the compression could be achieved by using previous messages (dynamic compression) because the similarity of the messages was often high (e.g. the address of sender and the receiver is the same) and there are algorithms which can benefit from this. It would be worthwile investigating this further.

9.3.2 Experiments of first attempts

First we tested the LZ77 algorithm. Since the dictionary contained only a few keywords as well as all the 256 ASCII characters, the compression ratio was over 100%.

In the second test the Huffman algorithm was used. The tree was built based on the frequency of occurrence of the characters in the message and then the message was compressed character by character. Since the tree was built dynamically, we had to attach information about the tree needed to decompress the message. This solution did not work because the compressed message was longer than the original. The compression ratio was over 150%, but for shorter messages the ratio was higher than 250%. This terrible (de)compression ratio clearly shows that the well-known compression algorithms cannot be used "as they are" in the case of data transfer with the SigComp layer; instead they have to be adapted or even redesigned.

Before investigating the other compression methods, we developed a good dictionary containing all the SIP keywords and the 256 ASCII characters. This dictionary consists of more than 600 keywords.

9.3.3 Creating a dictionary

The SIP messages are about 300-600 bytes long. A message can consist of all of the ASCII characters. The content can be divided into user data such as email, host name, . . . , and SIP instructions. We studied several SIP message flows which can be found on the Internet [36, 131], and used an idea in an article [31] where a static SIP dictionary was given. From this dataset we constructed a dictionary, which consists of 600 elements and can be divided into 5 parts:

1. Most probable special symbols ('=',' ', ...) (our elements)
2. Numbers (ten elements)

3. Alphanumeric characters (26 x 2 elements)
4. SIP instructions (∼500 elements)
5. Special characters (∼140 elements)

We should mention here that our algorithms use the same dictionary (the one presented here)and they only differ in the symbol coding algorithm.

9.3.4 Modification of the LZ77 algorithm

The first effectively working (i.e. compressing) encoder was an improvement of LZ77. The dictionary is of great importance in the case of the LZ77 algorithm, thus we used the revised dictionary. It resulted in some additional enhancements where the length could be represented just by 1 byte, while previously the length was 2 bytes. However, the real improvement was that we did not encode the short matches (1, 2, or 3 long) by LZ77 and these were forwarded un-coded. We introduced a special symbol to distinguish the encoded and the un-coded parts. The efficiency of the modified algorithm was still not as good as that of the Huffman and SubExponential compressors, but it was more efficient than the methods implemented previously implemented by us. The compression ratio was between 55% and 75%. Recurring parts in the message to be compressed further increased the efficiency, and an SIP message was one of these kinds of messages.

9.3.5 Prefix-free encoding

During prefix-free encoding we encoded numbers, so first the message had to be transformed into numeric data. The message was split into the keywords of the dictionary (only exact matchings were allowed), then the keywords were changed to the numbers assigned in advance of the keywords. After, we could use the two encodings described below.

The Rice encoding procedure was the first where we attained good results. During the encoding we encoded the numeric data transformed from the message, as described above. The parameter of the Rice-encoding (only one number) was chosen dynamically according to the message, and the numbers assigned to keywords were permutated constantly using a kind of "Move to Front" algorithm, to improve the efficiency of encoding. This permutation can be readily used in the decompression stage too. Only some (1 or 2) parameters were required to be attached to the compressed message, so it did not harm the overall efficiency. The compression ratio we achieved was 40-60% with this method, which was a significant improvement compared to the earlier methods applied.

The next test was with the SubExponential algorithm. It works like the Rice encoding procedure, but the length of codes here grew logarithmically depending on the numbers to be encoded, while in the previous case the growth was linear. The parameter (like that for Rice-encoding) was chosen

which depended on the message length and an improved version of the permutation was also used. The enhancement of this algorithm compared to the Rice encoding case was sometimes as high as 5% of the message length, but there were cases when it was negligible.

We developed the Synth algorithm as a simplified version of the Arithmetic encoder. Although its philosophy is rather different from that of the SubExponential encoder, the test results were very similar; sometimes the codes of keywords were practically the same.

Next we returned to the Huffman algorithm. Because of the large dictionary size it was impossible to attach the tree to the message, so we had to choose a solution that uses a predefined tree. This tree was independent of the message to be compressed, so it could be stored both at the sending and receiving endpoint in advance, thus we did not need to send the tree. Since the tree was not built based on the message, it was not optimal, but it approached the optimum quite well. The rate of approach naturally depends on how well we could estimate and define the tree, so the construction of the given tree was preceded by thorough testing. After having got the tree, the compression procedure was the same as before. We wanted to improve the compression ratio, hence first an adaptive version was implemented, but there was no significant improvement. Instead, we used a similar permutation method to that used in the Rice and SubExponential encoders.

9.3.6 Deflate and its modification

Deflate compression means the execution of two subsequent encodings. In the first step the message is compressed by the LZ77 algorithm, while in the second step the message is compressed again with the Huffman encoder.

We did things in the same way as before, but here we used our modified LZ77 instead of the original. After that, we did not compress the message immediately with Huffman. We separated the un-coded characters (including the special characters) and these were compressed with the SubExponential algorithm. The lengths were compressed with the SubExponential algorithm too, but independently of the type of characters. The MSBs of the positions (2 bytes) were compressed with the Huffman algorithm, while the LSBs were not compressed at all. We chose the SubExponential algorithm, because its efficiency was the same as that of Huffman encoder, but it was faster.

9.4 Evaluation

To understand the significance of the compression ratio and time for compression/decompression we have to analyze the mobile call setup. In the GSM system a typical call setup time is about 3.6 seconds [92]. The setup time of a SIP call is approximately 7.9 seconds [36]. We can calculate it with

a 140 ms RTT (Round Trip Time) and with a LinkSpeed of 9.6 kbps. In a typical SIP call according to an article [36] there are about 11 messages with an aggregated length of about 4,200 bytes. The SIP call setup time can be calculated from previous values with the help of the following formula [36]:

$$OneWayDelay = \frac{MessageSize[bits]}{LinkSpeed[bits/sec]} + \frac{RTT[sec]}{2}. \tag{9.7}$$

With this we obtained the following results:

TimeForTransmission	$\approx$ 3500 ms
TotalRTT	$\approx$ 630 ms
BearerSetup	$\approx$ 3000 ms
TotalDelay	$\approx$ 7130 ms.

The BearerSetup and the TotalRTT values could not be further reduced without expensive technical investment. The only solution is to decrease the TimeForTransmission portion with the help of compression. It is difficult to see how efficient the algorithms are because it strongly depends on the message type. We used the SIP message samples described in [36, 131] as the test set for our compressing algorithms. In the following part we will outline the advantages and disadvantages of the algorithms and compare each in turn.

9.4.1 Efficiency of compression

For the compression ratios, the reader should see Figure 9.5. First of all we observe that the compression ratio greatly depends on the message lengths (left figure). On the right hand side we notice that the LZ77 algorithm is still 10-20% worse than the others (Deflate, Huffman, ...) despite the fact that our modification substantially improved it . As we have already mentioned, it is difficult to choose the best of the three prefix-free encoding, therefore we will compare them to our Deflate algorithm. We can see that Deflate provides the best compression ratios. The second best are the prefix-free encoding algorithms, and the third best is the Huffman tree-based compression algorithm.

Our findings tell us that Deflate seems to be better because it is more efficient in the majority of the tests and – more importantly – it has two good features. One of them is that its ratio compression is not “too bad” even in the case of extreme messages, but here the prefix-free encoders perform very badly. The other feature is that if a message "can be compressed well", Deflate can indeed compress it much better than the others. This means that in some cases every encoder can attain a better ratio than that in the average case. However, the efficiency of prefix-free encoders increases only by 5%, while the efficiency of the Deflate encoder increases by 10-12%.

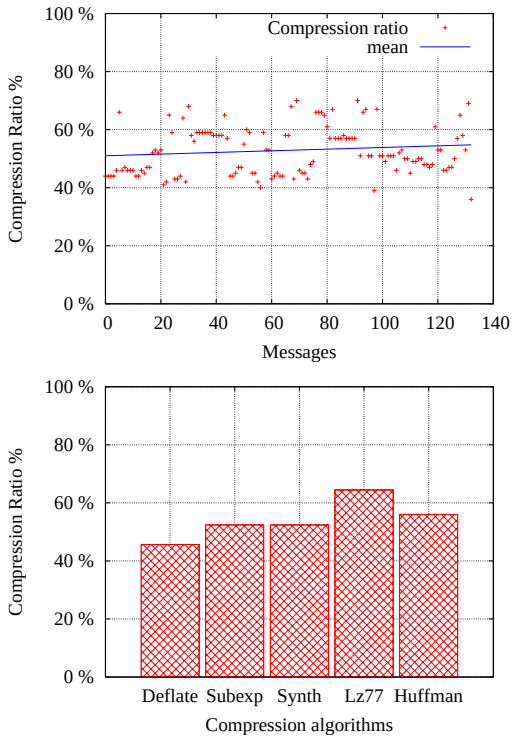


Figure 9.5: Compression ratios

9.4.2 Measuring the virtual time

We implemented both compressor and decompressor algorithms, but the measured running time cannot be used to estimate of the compression/decompression time because of the multitasking operating environment and different architectures involved. To estimate these values we have to use a theoretical approach. Both algorithms can be used on a mobile device and on a proxy server too. First of all we need some information about the central processing unit (CPU) of the mobile phone. According to an info sheet [51], CPUs in the today's mobile phones have a 100 MHz clock rate. In the case of decompression there is interpreted program execution (our byte code runs on a Universal Decompressor Virtual Machine), and we need to approximate the real clock rate by dividing the original clock rate by 10. From this we get the formulas (9.8) and (9.9). Here we did not consider the possibility of multiple instruction execution nor the possibility of complex instruction and hardware implementation of the virtual machine, so we would like to calculate the worst case. Now we need only the required CPU cycles of the different algorithms.

$$TimeOfTheDecompression \approx 10 \times \frac{NumberOfNeededCPUCycles[cycles]}{100MHz} \tag{9.8}$$

$$TimeOfTheCompression \approx \frac{NumberOfNeededCPUCycles[cycles]}{100MHz} \tag{9.9}$$

9.4.3 Time of compression

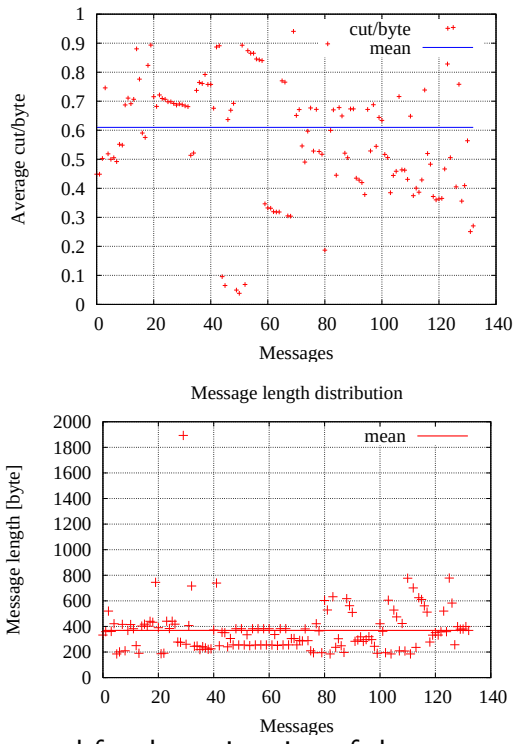


Figure 9.6: Parameters used for the estimation of the compression time of the prefix-free algorithms

The time of compression and the time of decompression can be more important than the compression ratio. Our goal is not to achieve the best compression ratio, but rather to achieve the minimal transmission time. It is evident that the time of compression depends on the message length, hence we will not emphasize it separately later on.

To estimate the time of compression for two compression algorithm groups (Deflate, LZ77, and prefix-free) we need different assumptions. First, let us calculate the time of compression for prefix-free algorithms. There are two main parts:

- The tokenization part — the algorithm divides the message into the largest possible tokens based on the dictionary. This part is the most computationally intensive. We can determine the mean of the average cut/byte with the help of Figure 9.7. The computer cycles needed

for the cut operations lie between 1 and 10. As a worst case we use 10 CPU cycles for each cut operation. For dictionary maintenance we need another 10 CPU cycles (move to front algorithm).

- The token encoding — this part is faster than the above. Let us assume 2 CPU cycles for each byte encoding.

With the help of previous assumptions we get the following results:

$$TimeOfTheCompression \approx \frac{Cut/Bytes \times MLength \times CutCost}{100Mhz} = 453\mu s \tag{9.10}$$

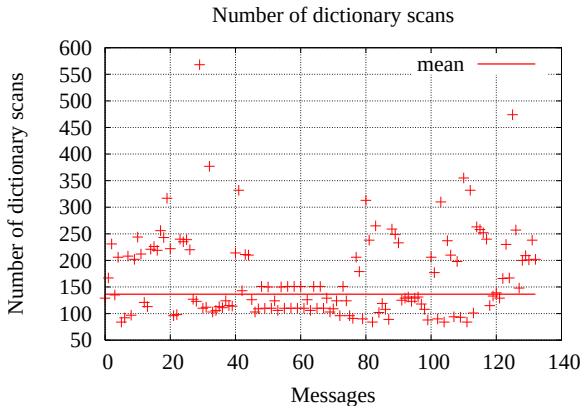


Figure 9.7: Parameters for the estimation of the compression time of the LZ77 and Deflate algorithms

The time of compression of the LZ77 and Deflate algorithms greatly depends on the size of the dictionary. In Figure 9.7 we see that the total number of dictionary scans is about 136. To have a dictionary we concatenate the message with the end of the dictionary. With the help of previous assumptions we get the following results:

$$TimeOfTheCompression \approx \frac{Scans/Bytes \times MLength \times (Dict.Lenght + MLength)}{100Mhz} = 1290\mu s \tag{9.11}$$

Here we may conclude that the prefix-free encoders are much faster than the LZ77 encoder and the Deflate algorithm on the compressor side.

9.4.4 Time of decompression

Since the decompression algorithm is executed by UDVM, it is very important to see know fast our algorithms are. The application determines a parameter that limits the cycles to be used to

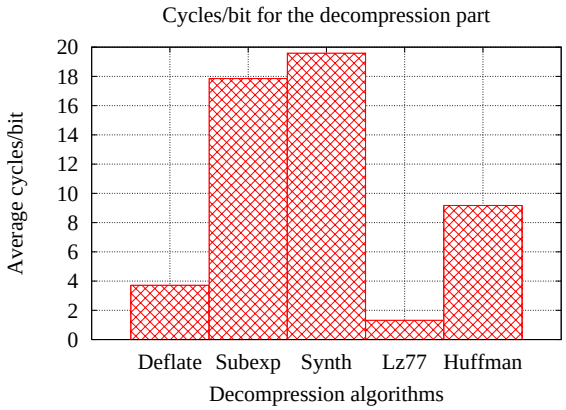


Figure 9.8: Cycles/bit for the decompression part

decompress the message (more precisely, the application limits the useable cycles/bit).

The LZ77 encoder is the fastest. Only 1 or 2 cycles/bit are needed for the decompression, hence it can be always used. The Deflate encoder is no less effecient because in its second step the prefix-free algorithm is used with fewer elements and without permutating them. The SubExponential encoder uses 18-19 cycles/bit; the Synth encoder uses 19-20 cycles/bit; while the Huffman encoder uses 8-9 cycles/bit. Nevertheless, these values are not too high because at least 16 can be used for decompression (the application parameter is at least 16 cycles/bit).

9.4.5 Memory

On the compression side there is no big difference in memory usage, because all the encoders use the dictionary and with the exception of LZ77 they use other data structures as well. None of the encoders mentioned here have a large memory usage.

On the decompression side it is curcial to know how much memory is used by the algorithms. Indeed, UDVM has a total memory of 64 Kbytes, but it could happen that less memory is available. In the following we will just summarize the memory requirements of the algorithms; but we should include the length of the uncompressed message in this figure because it is also required to be stored in the UDVM memory.

The LZ77 algorithm uses the least memory (3,5 Kbytes), although Deflate does not use much more (4,2 Kbytes). The Synth and the SubExponential decoders use 6,7-6,8 Kbytes of memory, while the Huffman decoder uses 12 Kbytes of memory. The latter is not surprising because it employs the Huffman tree to decompress the message.

Table 9.1: Summary of results

Algorithm	Comp. Ratio	Comp. T.	Decomp. T.	Transmission part of the Setup Time (without compression ≈ 3,5 s)
LZ77	64 %	1.55 ms	0.38 ms	2.2 s
Deflate	45.63 %	1,29 ms	1,09 ms	1.62 s
Synth	52.35 %	0.45 ms	5,78 ms	1.90 s
SubExp	52.38 %	0.45 ms	5.26 ms	1.89 s

9.5 Conclusions

As we mentioned previously, we could not find any articles in the literature which deal with SIP compression. So our task was first to analyze the SIP protocol and compression theory to find feasible approaches for data compression. Our present study is about the analysis, synthesis and comparison of several compressing algorithms in the SigComp layer. We developed a demonstration system which links two SIP user agents to each other and ensures the compression and decompression of the messages between them. The main parts of the system are the compressor / decompressor dispatcher, state handler, the compressor containing various encoder algorithms, the decompressor containing UDVM and a mnemonic-to-bytecode compiler. Several tests were performed to evaluate the performance of the algorithms (speed and memory usage) as well as the compression ratio; and, additionally, the conformance and robustness of our implementation.

As we saw, there is no optimal solution. Each compression algorithm can be good or bad depending on the criteria specified by the application. We should emphasize here that although the compression ratio is usually considered to be the most important, in the case of data transfer using the SigComp layer, special requirements exist and, as a consequence, the speed and the memory usage are more important than the accessible compression ratio.

The modified Deflate (run length encoding + context modeling) provides the best SIP call setup transfer time (1.62 s). However, this is the slowest solution on the compressor side, and in the case of many concurrent sessions (SIP proxy), this can cause a bottleneck. The modified LZ77 algorithm is the fastest on the decompressor side. And context modelling (i.e. prefix-free encoders) is the fastest on the compressor side. We should add that the algorithms are dictionary-based, which is optimal for messages with no special symbols. With the help of adaptive methods we can improve this optimality for messages with special symbols as well.

We achieved compressing ratios below 50% and decreased the total delay of SIP call setup by 1.5 seconds,and with dynamic compression (compressing message flow) this compression ratio could be

under 30% and the transmission delay about 1 s. Our experiments thus showed that SIP messages can be efficiently compressed. In a future investigation we would like to study dynamic compression and see how efficiently SIP can be integrated into the SigComp layer.

We think that our tests justified the philosophy of the protocol architecture; that is, let us allow the communicating parties to select the best method for themselves according to their requirements. The results shown in this thesis group are all the results of the author and were published in research paper [FBSS03].

Theses:

Thesis 9 *The classical compression algorithms without modification are not applicable for SIP compression.*

Thesis 10 *The modified Deflate (run length encoding + context modeling) algorithm is the best SIP when the call setup delay is important.*

Thesis 11 *The modified LZ77 algorithm is the fastest on the decompressor side. Context modelling (i.e. prefix-free encoders) is the fastest on the compressor side.*

10
Scalable storage

In today's hectic world time is money and so is information. This is especially true nowadays with customer data from e-business and the huge amount of logistic and scientific data, which may both be worth their weight in gold. The amount of data is also increasing sharply, and the average storage capacity you get for your money is skyrocketing. The storage of several hundred GBytes is achievable for everyone. One might argue that today's storage capacity is just following the trends and there is enough cheap storage to meet the increasing demand.

Unfortunately, the total cost of ownership is also increasing sharply with the amount of maintained data. In a typical company there are several file servers which provide the necessary storage capacity and there are many tape libraries for archiving the contents. If the storage needs grow the company can purchase a new hard disk or a new server. To have a reliable system there is usually replication between the dedicated servers. The disk drives are organized in RAID arrays, typically RAID 1+0 or RAID 5 [94]. This solution is not scalable enough for today's Internet applications where there can be huge fluctuations in demand. Failsafe behaviour versus effective storage capacity ratio is not optimal because of mirroring. Management is the other weak point of this system. This was why the Storage Area Network was designed. In a typical SAN there are several storage arrays that are connected via a dedicated network. The storage arrays typically contain some ten to sixty hard disks. To protect the data from hard disk failure these disks are organized into RAID 0, 1, 5 arrays. Safeguard from two or more hard disk failures is very costly because of mirroring. In larger systems it is vital to protect the data against storage array failure; hence the storage arrays are duplicated and connected by SAN

switches. The servers are connected to this network via their fibre channel interfaces and provide a 2 GBit/s transfer capability. The scaling of this system is achieved by adding new hard disks to arrays, or moving the partition boundaries. The price of SAN components is high compared to typical network components and servers, and the storage usage failure toleration ratio is not so optimal.

We would like to present a cheaper solution to this problem. A typical PC now has a huge computing and storage capacity. It is not unusual to find more than 100 GBytes of storage capacity, over 500 MBytes of RAM and two GHz or more CPU clock frequency in a desktop PC. It seems that these parameters are constantly increasing. A typical installation of an operating system and the software required does not consume more than ten to fifteen GBytes. The rest of the storage space is unused. A typical medium-sized company has more than 20 PCs. A university or research lab usually has over two hundred PCs. In this case the storage capacity that is wasted may be several TBytes in total. So it would great if we could utilize this untapped storage capacity. In order to solve the above-mentioned problem we decided to design and implement LanStore with the following design assumptions:

- It is highly distributed and without any central server functionality.
- It has a low server load. We would like to utilize the storage capacity of desktop machines; these machines are used when our software runs in the background.
- It is optimized for LAN. The use of multicast and a special UDP-based protocol is acceptable.
- It has effective network usage. We designed and implemented a simplified UDP-based flow control protocol.
- It is self-organizing and self-tuning. We used a multicast-based vote solution to implement the so-called 'Group Intelligence'. Here the environment is highly fluid. The desktop machines are restarted frequently compared to dedicated servers.
- It is a file-based solution. For effective caching we chose file-based storage instead of a block-based one. [70]
- It has campus, research laboratory-type file system usage. Also, file write collisions are rare [70].
- It has an optimal storage consumption failure survival ratio. As a first approach we opted for Reed-Solomon encoding for data redundancy.

10.1 Related work

Distributing the contents among storage blocks is by no means a new idea. The oldest and the most popular technique is the RAID (Redundant Array of Independent Discs) technique [94]. It uses two basic data distributing solutions called stripes and mirroring. The first algorithm uses XOR parity data slices for correcting only one error, while the second one can be used several times to achieve the necessary error correcting level, but the storage efficiency then sharply decreases. RAID is typically used for computers with several hard disks inside. The Zebra [37] file system took the idea of striping from RAID, but instead of distributing the data among hard disks it distributes the data among storage servers. To effectively use the network bandwidth it uses per client striping instead of per file striping. The weak point of this solution is its single error correcting capability. Petal [79] uses striping without redundancy and mirroring as a type of data distribution. One can define block level virtual disks with the aid of a low-level interface. There are special server functions which translate the addresses used on a virtual disk to a physical machine and disk addresses. It uses a heartbeat backbone to provide the so-called "liveness" property. A distributed consensus is achieved by using Leslie Lamport's Paxos [77] algorithm and the goal of the Pasis [57] project was to create a solution for building a survivable data storage that was as simple as possible. Here there is a thick client and thin servers. The only functionality implemented in servers is the data store which can be implemented as a simple file share, except that all this functionality is implemented on the client side. For the object name to physical location mapping, a directory server is used. In a later article [56] the authors of the Pasis framework define a new approach for handling Byzantine[9]-type failures. In this solution the correction of failed storage nodes is a client task; there is no background process for consistency maintenance. This solution does not utilize the computing power of server nodes. Self*-store [53] is based on Pasis, its goal being to create a safe storage where, for a specified duration, there is no chance of data erasure. If the logfiles were stored in the Self*-store then the intruders would not be able to erase their footprints. OceanStore [110] defines a global scale storage system on a multicast overlay network. They use Tapestry [5] for object naming and locating. To achieve data redundancy they use both erasure codes and mirroring. There are several defined classes of storage nodes with different responsibilities. For example the inner ring members have the task of data redundancy handling, but this solution is unsuitable in a laboratory where the storage nodes are desktop machines and they cannot tolerate a heavy processor load from a background process. FAB [109] defines a storage system with a block level interface. The clients use SCSI commands for data manipulation whose implementation uses the thin client and thick server paradigm. This solution is unsuitable in an office or laboratory.

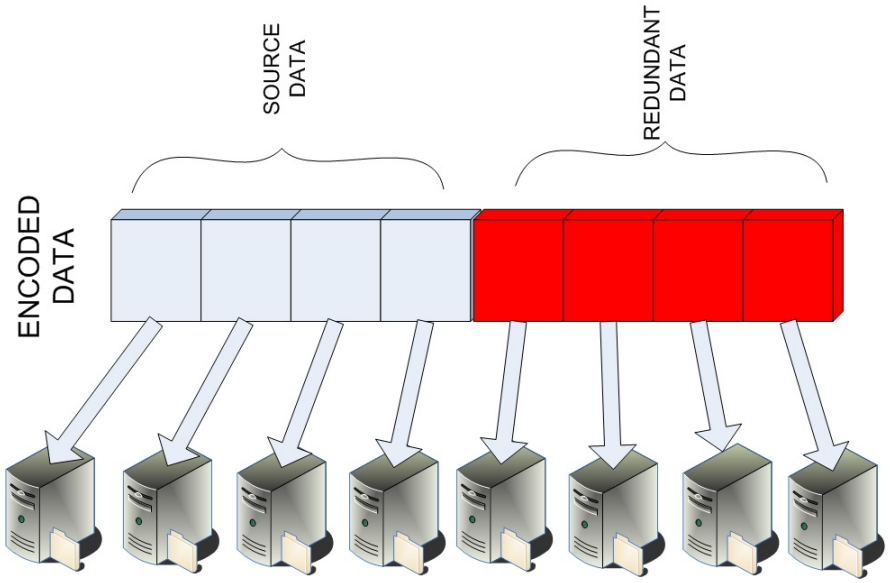


Figure 10.1: Basic idea

10.2 The architecture

Before going into detail let us see the high level workings of LanStore. As we mentioned, before the main design goal was to gather the empty storage capacity into a virtual storage unit. To utilize in a uniform way the storage capacity of the member nodes, we divided the files into equal fragments. In this way every storage node has the same number of stored data fragments. We would like to collect the free space from PCs in computer laboratories, classrooms, and so on. There is a high probability that one or more machines will be rebooted or turned off. We need data redundancy to correct the data which is stored on these machines. We will use forward error correcting codes (FECs) for error correcting. With the help of these algorithms we can create n data fragments for m original data fragments. This means that we can reconstruct n failing data fragments. This process is shown in Figure 10.1. The consistency among modules is provided by a voting algorithm. If there are a critical number of working data nodes the remaining nodes may be reconstructed. Our solution is transaction based. At the end of a transaction a vote is taken and any changes are written to a permanent storage unit when the majority of nodes agree on the next common state. If there is no majority acceptance of the new state, the transaction will roll back. After the changes are written into a permanent storage, a second vote is taken of the result. If there is a successful majority vote the whole task will be marked as fulfilled; if there is no successful majority result the first and the second transactions will roll back.

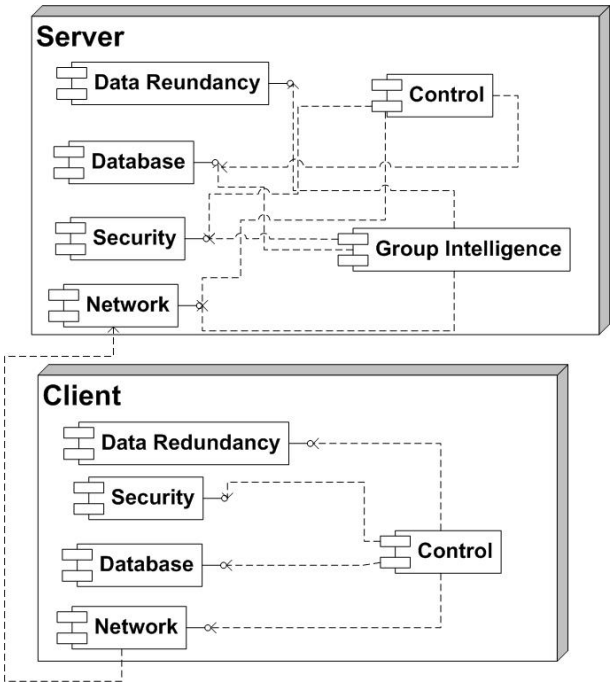
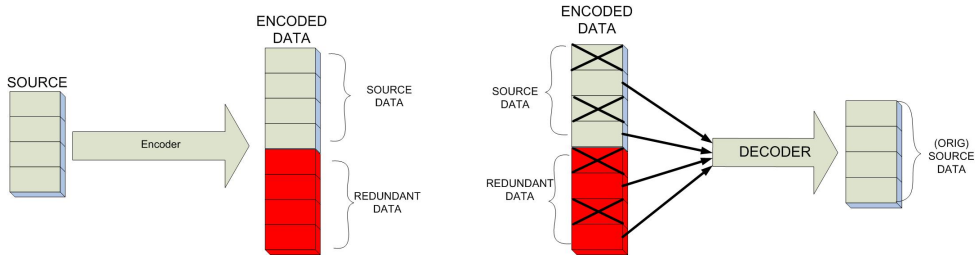


Figure 10.2: Modules

In our system the file is the basic data unit. We designed the file store for campus and research laboratory usage where file-based caching could be much more effective than block-based caching [70]. The files are identified with the aid of the hash of their contents. With this solution we never store the same file twice. If someone tries to upload a file that already exists in our storage system, it creates a new link to the existing file. In the case of a modification, the storage uses versioning to handle the modifications. Our application is divided into independent modules. This design pattern provides an easy-to-maintain and robust code, where each module can be replaced by another one using interfaces. The necessary functionality groups of our software provide us with natural borders among modules.

The modules are the following:

- The Data redundancy module
- The Network module
- The Data persistence module
- The Security module
- The Group intelligence module



(a) Encoding (b) Decoding

Figure 10.3: Encoding/Decoding

- The Application logic module
- The GUI module

Figure 10.2 shows the communication path between the modules. The control module is the core of our application; it uses the services provided by other modules. It is singleton, while every other module is thread safe. We may find that some modules in the client and server sides are the same, which contradicts our goal of developing an application with a fat client and thin server. During normal functioning the server does not use its Data Redundancy module. It only stores, sends the necessary data fragments and maintains its state with the help of the Group Intelligence module. We need the Data Redundancy module only for heavy data migration when every server helps a new or old server in an inconsistent state to achieve the consistent state.

10.3 Data redundancy module

The task of this module is to provide the necessary data redundancy for error correction. Several approaches are available in the literature. The most popular one is that of data mirroring. This is an easy- to-use and implementable technique with low processing overheads, but we pay the price on the storage consumption side. The creation of data parity blocks is another popular way, but apart from its optimal storage consumption this method can correct only one error at a time. This is a big drawback. For our goal a special class of the forward error correcting codes (FECs), the so-called erasure codes, provide the best solution. Since we can detect failing data, we only have erasure errors. In the case of FECs one can select the required redundancy level and the algorithm generates the necessary error correcting data blocks for the existing data blocks (see Figure 10.3). If a data block fails, it can be calculated from the remaining data and error correcting blocks. There are two types of FECs: codes with guaranteed error correcting capabilities and codes which have an error correcting capability with a given probability. We opted for the first code family because of its guaranteed error

correcting capability. The price, however, is the processing overheads which depend on the selected error correction capability. This is one or two magnitudes higher than that for the second case. We chose a special case of the Bose-Chaudhuri codes called the Reed-Solomon [107] code. The basic theory for this is quite straightforward: we have n data blocks and we need m data blocks to correct fewer than m erasure errors. To produce m data blocks we require a special equation system where every partial matrix is invertible. To produce such an equation system the Reed-Solomon approach makes use of the Vandermonde matrix. The Galois field is used as the space where the operations are performed. With this solution we replace the complex calculation-intensive operations by lookup tables. Here we use the Luigi Rizzo [107] implementation of the Reed-Solomon code. The module divides the processed files into 64 KByte long stripes and calculates redundancy data for these slices. These stripes form the basic unit of the versioning system.

10.4 Multicast flow control

Our software is designed to run in a LAN environment. Most modern LANs are switched and there is practically a full mesh among network nodes. The key feature of such a network is that the bottleneck is on the source side or on the destination side; the network itself does not contain bottleneck nodes. TCP was designed and optimized for situations where the network is a black box and we can detect the available bandwidth only with the help of packet loss. There is an optimal windowing algorithm [21], but this is not optimal when there is more knowledge and we can use a multicast protocol. We have complete knowledge of both sides of the communication channel, so it is plausible to use a flow control mechanism based on this. We designed a simple flow control mechanism that is capable of handling both multicast and unicast traffic. UDP here was used as a base and we added a simple signaling mechanism. Prior to each data manipulation process a transaction identifier is created by the client from the hash of the manipulated file and the public key of the client, this ID being unique to the whole system. At the same time only one client manipulates a file. Our multicast flow control mechanism has two working modes, both modes utilizing the error correcting capability of our solution. In this way we can strike a balance between processor occupation and network transfer capability. The download mode operates during data transfer from a group of servers to a client. The upload mode operates during the data transfer from a client to a group of servers. In the following we will describe these modes.

In the download mode the client receives the file segments from servers and then stores these fragments in the input queue. If there are sufficient fragments for error correction (Figure 10.5, line 6) the client immediately starts the error correcting process. When it finishes the error correction, an alert is sent to the controller and it sets the processed bit for the processed stripe (Figure 10.5, lines

```
Download mode:
1. Receive(fragment, stripeId, from)
2. IF(stripe is not yet processed)
3.  StoreFragmentInQueue()
4. CheckQueue()
5. ELSE
6.  Drop(fragment)
7. END IF
8. IF(the Queue occupation is over 20%)
9.  SendFlowControlInformation()
10. END IF
```

Figure 10.4: Download mode

```
CheckQueue function:
1. IF(there are more than N data fragments for the same stripe)
2.  IF(we have every data fragments)
3.    SendAlertToControler()
4.    SetTheProcessedFlag(stripeId)
5.  ELSE
6.    StartErrorCorretion(stripeId)
7.  END IF
8. END IF
```

Figure 10.5: Queue handling

3&4). Additional fragments for the processed stripes are dropped. With this solution we can avoid the situation where bottleneck nodes slow down the data transfer rate, and we can tolerate transparently the failure of nodes below a critical number. In the upload mode our task is similar, namely that of tolerating the node failures and avoiding the situation where several slow nodes decrease the speed of the whole upload process. In this case after the first control packets the client starts sending the data fragments to different nodes as unicast UDP packets. When a storage node notices that the free space of its input queues is below 80%, it sends a control packet to uploading clients with a preferable transfer rate. The client has the responsibility of deciding whether it will accept the request or continue the upload with a higher speed. The decision of the client is based on responses from other storage nodes. It selects a speed which is acceptable for a value above a critical number of storage nodes. The rest of the nodes will be corrected with the help of the Consistency process, which is a part of the group intelligence.

10.5 Group intelligence module

In a distributed system this module plays a very important role. Its main task is to provide consistency, meaning a consistent state and consistent databases. In an ideal system where there are no failures this is not a hard task, but such difficulties arise when we have a real system. In the real world there is no algorithm that provides guaranteed consistency. To be able to handle this situation we shall define the following model of reality:

- The participants in the group management protocol can reboot or switch off at any time.
- Recorded data can never be overwritten.
- The messages must be delivered without delay or they will be lost.

With these constraints this module has:

- A voting-based algorithm for sequence upload verification
- A voting-based algorithm for file modifying finalization
- A voting-based algorithm for designated node selection
- Management of the correcting process of failed nodes

The voting algorithm is based on one by Leslie Lamport called Paxos [77]. Every server node maintains a history database that contains the successfully finished instructions. A data modification or upload is a sequence of stripe uploads which are a sequence of data fragment uploads. After every stripe upload a vote is taken of its success. If it was successful this fact is placed in the history database. After every data modification transaction (sequence of stripe uploads) a vote is taken of the success of the transaction. The success of a transaction really means that every sequence upload vote was successful. If a transaction was successful then every node erases the temporality signaling flag of the modified file. After this is carried out, the new version of the file is the latest version.

A designated node is important when the group of storage nodes sends messages to the client. This happens when a client asks for the new file list and about the success of a file modification. The load of the processor, the occupation of the memory and the stability of the node are vital properties during the designated storage node election process. The designated nodes are changed after a few dozen transactions. The correction of failed nodes is handled collectively; each consistent storage node is responsible for a stripe. The sequence of tasks needed to correct it is calculated using the data difference between the local history table and the globally accepted one. To calculate the required data fragment these nodes act as clients. With this method we can achieve a relatively fast self-correcting capability of the group without imposing a high load on any given node. There are

so-called synchronization points where a part of every history table in the system is the same. After reaching several such points the old records are deleted from the history table.

10.5.1 Our Paxos implementation

Functionality is provided by three abstractions: Leader, Consensus algorithm, Learner. From a high level point of view the system works as follows. The clients send instructions to a leader. This leader carries out a three-phase transaction on the participating nodes and sends the results to the client. Now we will describe the algorithm and a detailed description of our implementation. Initially, during the implementation phase of the classic Paxos algorithm we had to solve the following problems:

- Message ordering: The purpose of the leader abstraction is to serialize the incoming requests. As we saw earlier this task can be done in a distributed manner (with logical timestamps and so on), but these solutions are more costly and are less reliable than the single leader solution. One could argue that the single leader incorporates a single point of failure into system. This is true, but as the leader does not have persistent data it can be easily replaced by a live substitute.

- Leader election: As a communication medium between the Leader and the participating nodes, the Instructions multicast channel is used. During idle periods, the Leader periodically multicasts a beacon packet that contains the number label of the latest instruction. Based on our experience in other areas, we chose to set this period to 10 seconds. During active periods these packets contain Paxos instructions (Propose, Accept, Decide). Failure detection is achieved by timeouts. If there is no traffic on this channel for three times the beacon period (30 seconds), the clients will submit a LeaderSelect frame that contains their stability properties (the greatest message serial known by this node, the number of restarts, the duration of the longest stable period). Each node compares the received values with its values and if it discovers that its values were better (in the case of equal values the greater IP number is chosen) it will wait for a random period between 0 and 15 seconds and start sending beacon packets. If a node thinks that it has the best values but receives beacon packets, it will accept the new leader. With these settings a Leader change will last at most 45 seconds.

- Learning the actual leader: There is a Leader channel where the leader submits the beacon every 30 seconds. This channel is intended for clients to determine the actual leader. The clients send the data items to be stored to the leader using a TCP connection.

- Monotony maintenance: The leader node retransmits the messages from the clients to the Instructions multicast channel and with these values assigns a global number G and local

number N to the messages. Local numbers are interesting only when there are two or more leaders. These numbers should be unique among the leaders so they are constructed as follows: IP address+ N*232. For every submitted message G is increased and N is reset to 0. G and N are included in the beacon packets as well.

Every node in the distributed system is subscribed to the Instructions multicast channel. For each different global number there will be a separate "Synod" protocol that guarantees consistency among the nodes. It works as follows:

Phase 1. The leader selects a global G and a local number N for the instruction and sends it as a proposal for the nodes subscribed to the Instructions channel. This is the so-called Prepare request. If a receiving node receives a Prepare request it checks to see whether it is able to accept it. If the last accepted request has a global number which equals the received global number and the local number is less smaller than that of the current request then it responds with a reject answer; otherwise it will send a prepare accept response. Both responses contain the last accepted request and the also the number of this request.

Phase 2. If the leader receives a response for its propose request from the majority of the nodes, then it selects the latest accepted request, or if there was no request previously then it2 uses its own request and sends an accept request to the Instructions channel. In the case of insufficient responses or a reject answer it will increase the local number and submit the prepare request again. If there are insufficient responses after the fifth unsuccessful round it will stop the process and send an unsuccessful message to the clients. If it gets one or more reject answers it will increase the N value and send the message again. After five unsuccessful turns it will increase the value of G to the maximal value reported by the clients plus one received in the reject messages. If it is unsuccessful then it will report this to the client. This situation can happen only when there are multiple leaders and they all are functioning for a longer period of time. But this may happen only in very special circumstances. It is quite rare. The node receiving an accept request checks the local number of the request, and if it is greater than the last accepted one or there was no such G then it accepts the request and sends an accepted message to the leader. Otherwise a reject response is sent with the N value and maximal known G value.

Phase 3. After receiving sufficient accepted messages the Leader sends a Decided message to the Instructions multicast channel. The node that receives the Decided message will insert the Decided values into its Decided values storage. The timeout for each phase is 20 seconds. If the number of received accept messages was less smaller than the previously defined majority value it will try sending

the accept request again. If it fails five times it will send this result to the client and stop the process. In the case of a reject message it will follow the process described in Phase 2 and restart Phase 1. If the chosen value was not the value originally sent by client, then the Leader will repeat the whole process until the decided value and the accepted value coincide. This situation may occur if the G value known by the leader is less than the greatest G value in the whole system.

A detailed description of this algorithm can be found in [77] and [78]. We implemented the Paxos algorithm using several optimizations to achieve better response times:

For the system to progress we need the majority of nodes to be live. It may happen that in a fluctuating system, the majority of nodes are always present but are constantly changing. For example the prepare request is received by node A, then node A restarts and node B finishes its restarting process. So node B will only receive an accept request. The classic Paxos algorithm recommends rejecting this message. But with this solution it can happen that we have to replay the whole propose/accept procedure. Instead of this we suggest the following. If a node receives an accept request without previously receiving a propose request it shall answer this request. If it disagrees with the value suggested by the accept request it shall handle the accept request as a propose request; if it agrees with the received value then it shall handle the accept request as a propose and accept request. With this modification we did not change the durability of the algorithm, but in some cases we reduced the required number of message exchange actions from six to two.

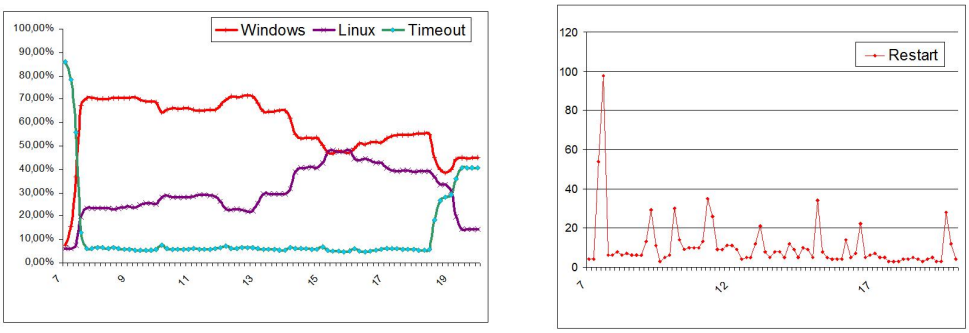
Change of membership: The participating nodes maintain two lists of instructions. In the "Client list" the data items submitted by the clients are stored, while the "System list" contains the instructions for system maintenance. The handling change of membership is solved by these special instructions, which are treated the same way as instructions received from clients.

Message optimization: A Leader may incorporate an arbitrary number of Paxos messages with different G values into one submitted packet. The Decide packets may be piggybacked to Accept packets. The prepare packets are only required during the start of a longer stable period. With these optimizations we then need only one message per transaction during stable periods. The details of these optimizations were described in part in two published papers ([77], [78]).

Slow/Fast query: A client may learn the chosen values in a fast or slow way. The fast way is to query the adjacent node about its list of decided values. The slow way is to perform a distributed query of the missing values. This query is submitted to the Instructions multicast channel. The distributed query contains the number label of the last known decision. The nodes receiving the query will respond to and return the accepted values. The client will summarize the answers and in

```
Phase1:
     Server:
        Var ReceivedRequest([G[N,V]], Iteration=0
        SendPropose(Nx232,G)
     Node:
        Var ReceivedProposes [G[S,V]]
        ReceivePropose(S,G){
        IF(G not known)
          SendAcceptPropose()
        ELSE IF (Smax <S)
          SendAcceptPropose(Sloc,Svalue)
        ELSE
          SendRejectPropose(G,Smax,Gmax)
        }
Phase2:
     Server:
        IF (ReceivedAcceptPropose > Memb/2)
           IF(MAX(S) != 0)
             SendAcceptReq(G,Sloc,Svalue)
           ELSE
             SendAcceptReq(G,Sloc,V)
        ELSE
           IF(N<5)
             N=N+1
             GOTO Phase1.
           ELSE
             REPORT ERROR
     Node:
        IF(G not known)
           SendAcceptReq()
        ELSE IF (Smax <S)
           SendAcceptReq ()
        ELSE
           SendRejectReq(G,Smax,Gmax)
Phase3:
     Server:
        IF NUM(ReceivedAcceptReq > Memb/2)
          SendDecide(G,V)
        ELSE
          IF(N<5)
            N = N+1
            GOTO Phase 1
        IF(Sv != V)
          G = G+1
          GOTO Phase 1
        ELSE
          SendSuccess()
```

Figure 10.6: Algorithm



(a) Operating system percentage / hours (2006.02.20)

(b) Number of restarts every 10 minutes (2006.02.14)

Figure 10.7: Churn in the laboratory

the case of unknown new decisions it will send a decide message to the Instructions multicast channel to assist the progress of the whole system.

10.5.2 Churn in a laboratory

Our university has a computer science laboratory with 204 PCs. Students can either use the Windows o.s. or Linux o.s. from 8 am to 8 pm, and they can switch between the operating systems whenever they want. We measured the machine availability by pinging these machines every minute for 3 weeks between February 6 and February 25 in 2006. Based on the TTL value of the response, we were able to detect not only the failures but the type of operating system used as well. We found that in a week the mean number of the online Windows workstations was always above the critical 50%. Figure 10.7 shows the same statistics, but now for a particular day. We observe that during the day except for a short period the number of online windows machines was above the critical level. The difference was about 10%. In the next figure the number of restarts is shown for another day. We see that there are situations where over 10% of the machines were restarted. In such cases it may happen that during a transaction more than 50% of the windows machines are online, but the ones that are running may vary. From these measurements we may conclude that for a reliable and *liveness* system we have to take into account these special situations as well.

10.5.3 Validation of the Paxos implementation

We tested our implementation in different circumstances to demonstrate that the single leader role does not affect its stability. To be able to simulate different network conditions we developed a

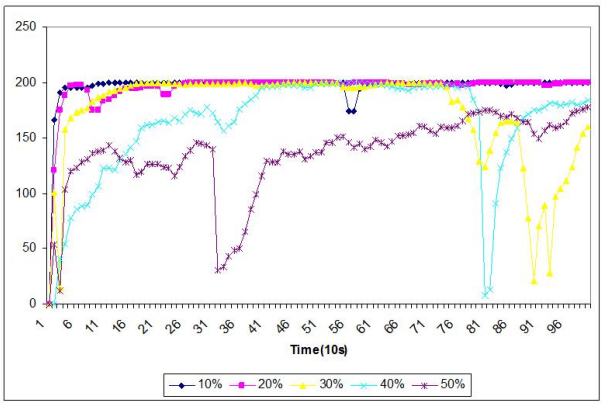


Figure 10.8: Number of threads that know the good leader at the same time

simulation framework where each machine was simulated with a separate thread. With the help of this solution we were able to fine-tune the machine restart probability values. In the following we will present our results on the stability of the leader election process. During the experiment we simulated 200 PCs with a restarting probability of 10% to 50% . In Figure 10.8 we can see that the system converged in a very fast manner in the case of a low restarting probability. If we raise the restarting probability the system also converges, but in this case the convergence is slower, and it contains more peaks.

10.6 Security

The security module has the task of providing data integrity, user and node authentication and access control. We store the digital certificates of nodes and users in the central database; the MD5 hash and the Windows SID are stored here too. We use the existing Kerberos infrastructure for authentication whenever it is available. When there is no such infrastructure then we provide a simple asymmetric encryption-based authentication infrastructure. The data integrity of messages is safeguarded by digitally signing them with the sender's private key.

10.7 Data Storage

The data storage module is responsible for data persistence and it has to maintain the history of conducted processes. The stored data can be divided into two main groups; the information which must be globally consistent and the information which is of local importance (Figure 10.9). The Group Intelligence module maintains the consistency of globally important information.

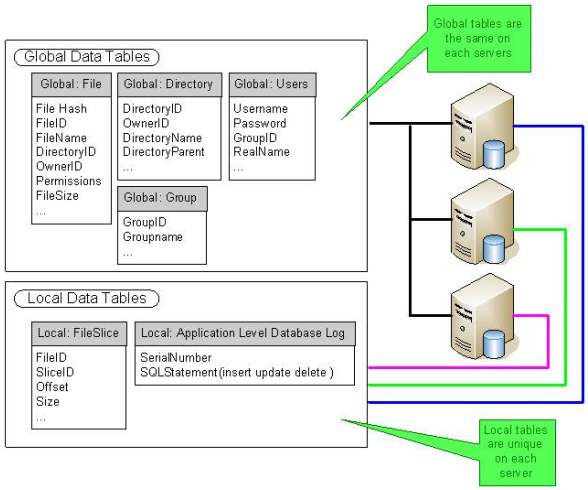


Figure 10.9: Data model

Using the model we store the following information:

- Metadata about data such as the file name, path and access control lists.
- The data that is needed for the correct working of our system e.g. users, nodes and certificates.
- The file fragments which have to be stored.
- A history of the processed instructions.

Every data type has its own prpoerties and therefore we chose different solutions for persistence. Meta data, infrastructure data, and histories are stored in a lightweight relation database. The size of this database never exceeds some 10 Mbytes, but the fragments can be several hundred MBytes. We tested the handling of large objects in the current databases. From the experiments we may conclude that the conventional file system has a speed that is about ten times faster for file fragments than that for the current database solutions.

10.8 Implementation

We opted for the Windows platform because of its widespread use in offices and university laboratories. Because they are well integrated in the Windows platform, the .NET framework and the C# language were also chosen. For example it was straightforward to check the infrastructure and the computing power of the hosting PC for leader election with the help of the Windows Management Instrumentation service. Another reason for using the .NET platform and managed code against the unmanaged C or C++ code was the short development cycle. Five graduate students worked for a year developing

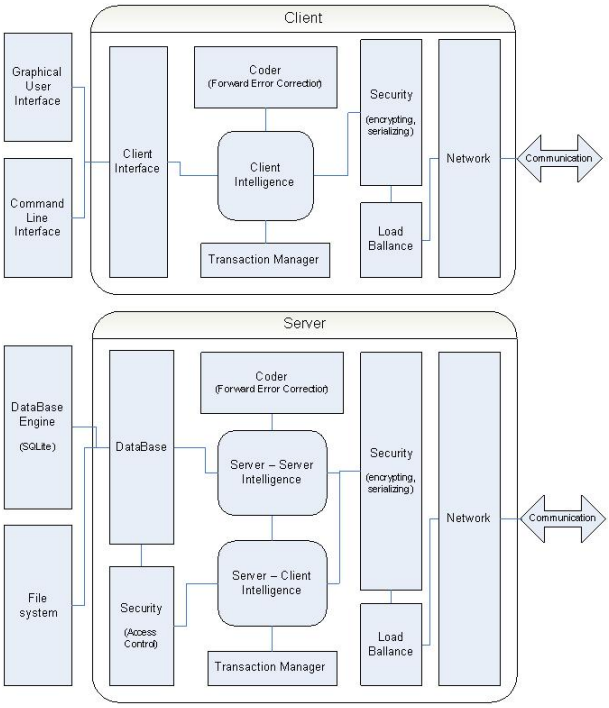


Figure 10.10: Our implementation

the software, which is now in the alpha state. It has currently over 20,000 lines of code. Figure 10.10 shows the detailed architecture. On the client side there are two threads, namely the Network module and the Client integration module. The network module has the task of capturing incoming packets and storing it in a synchronized queue. We designed this module to be as simple as possible so as to be able to capture every packet. The Client integration node processes the packets from the common synchronized queue with the assistance of helper classes. If the queue is empty then the thread will go into the wait state. In this state the network module can wake it up with a pulse signal. In the case of file upload the GUI uses asynchronous method calls for each storage server. In this way outgoing traffic is handled in parallel. As the network module does not inspect the contents of the package and the packages could be encrypted with only one thread, the original client integration thread for handling the incoming will decode the packets and, if needed, wake up the appropriate sender thread for handling the output traffic. The server side has a similar architecture, but instead of a GUI there is a database engine and a Server-Server intelligence module. The following four threads are always running: the Network, the Server-Server, the Server-Client, and the Hello thread. The first three threads work the same way here as on the client side. However, there are two queues; one for Server-Server and one for Server-Client module. The Network module makes a decision based

CPU Clock Frequency (GHz)	N	K	Throughput (MBit/s)
1	64	32	40
2	64	32	80
3	64	32	120
3	200	100	38.4

Table 10.1: The Reed-Solomon performance in our setup

on the type of destination address of the incoming packet for selecting the appropriate queue. The Hello thread has the simple task of periodically sending hello packets. These packets act as keep-alive packets. Owing to its speed, small size and easy-to-deploy capabilities, SQLLite was selected as the database engine, but it has no transaction handling capabilities. If one attempts more than one writing process simultaneously, it throws an exception. To avoid this, we used the .NET frameworks ReaderLock solution to achieve a serial access of this resource. As we mentioned earlier, we used the FEC encoder implemented by Luigi Rizzo [107]. Here we treat it as a native code.

10.9 Evaluation

The raw encoding capacity with Reed-Solomon encoding was measured first. The results are shown in Table 10.1. We may conclude that the currently used processors produce a useable throughput for 64/32 (64 nodes, and out of these 32 contain error correcting information). To test the performance we used a laboratory with sixteen PCs, each having P4 3 Ghz processors, 1 GByte of RAM and a 100 MBit/s network adapter, and for debugging we used virtual PCs. We measured the throughput in different scenarios. Even in a larger configuration when there were 16 servers and we used a 16/8 redundancy scheme, the 100 MBit/s network bandwidth created a bottleneck. The processor utilization was only 20% on the client side, and less than 1% on the server side. The above-mentioned measurements give a picture only about the raw coding capacity of a typical PC. Although this process is the most time-consuming part of the whole transaction, the remaining task could add significant delays. To be able to compare our solution with existing systems, we tested our framework in different scenarios. One of the most accepted methods of file system testing is the Andrew benchmark [54], which was created to measure the efficiency of the Andrew file system. This benchmark contains the following measurements:

- MakeDir
- Copy
- ScanDir

- ReadAll
- Compile

It measures the time needed for these tasks. Among these popular tasks, the size of the manipulated files is important. An article [82] estimates the distribution of file sizes of the UNIX file system as a Pareto distribution with parameters a=1.05 and k=3800. In another paper [26] it was demonstrated that the Windows file system file-length distribution could be modelled with the help of a lognormal distribution and a tail with a two-step lognormal distribution. As a simple, but appropriate solution we chose the Pareto distribution to model the file-size distribution of user homes. Currently our system is accessible only through the GUI provided. We do not provide an API, so we cannot use the original Andrew benchmark script. In these circumstances we did the following and then took measurements: we created an application which generates files with the length of a Pareto [82] distribution and the depth of its directory path has a linear distribution. Each character inside the files is generated with a linear random distribution. We uploaded and downloaded the generated file/directory set with the help of the GUI. We used the Windows SMB file share as a comparison partner. A test network was set up with 10 PCs, each having P4 3 Ghz processors, 1 GByte of RAM and a 100 MBit/s network adapter connected via a HP4108 switch functioning as server nodes and a similar PC to the client node. The redundancy ratio was set to 7/3, so for every seven original data items three error correction items were generated. The following tasks were measured on the LanStore and on a Windows share which was one of the server nodes:

1. The delay of directory creation (a), and deletion (b) in seconds, with 615 randomly generated directories, with depth and name space of a random linear distribution. We executed this task on LanStore and on a Windows file-share system.

2. The delay of file upload (c) and download (d) in seconds and the throughput in MByte/second with 200 randomly generated files with a Pareto size distribution(a=1.05, k= 3800) and with a random hierarchy. The aggregate size of these files was 4.08 Mbyte.

The results we obtained are shown in Table 10.2. From these results we may conclude that for small files our system is about two magnitudes slower than the currently used network file systems. The reasons for this lie in the distributed nature of our system. In the current implementation each operation is handled in separated transactions and after each transaction a vote is taken of the success or failure of the transaction. As we saw with small files and with administrative tasks like a directory tree manipulation, these time overheads can be longer than that of the whole file upload. We can correct this problem by batch processing the operations. When we upload a directory we can then assign a transaction for the whole process instead of managing every single operation as a

	Lanstore		Windows file share	
	Delay	Throughput	Delay	Throughput
a	353	-	5.3	-
b	116	-	3.8	-
c	213	0,02	3.5	1,25
d	53	0,08	6.1	0,7
e	262	4.02	144	7,32
f	240	4.39	104	8,5

Table 10.2: Results

transaction. To test the framework as a video archive, we had to take measurements using different file size distributions. The video files are in most cases larger than normal files, so we used the value of 3,800,000 for k. With this value we generated 75 files with an aggregated size of 1.03 GBytes and the directory hierarchy was randomly generated. The test bench was the same as in the previous measurements. The results we got for file upload and file download are shown in rows (e) and (f) of Table 10.2. We can see that with larger files our solution provides a delay and throughput comparable to traditional network file systems. With batch processing this result can be further improved. In the case of a stabile environment we can achieve a higher throughput than tradition file systems by sending the error-correcting data fragments only when they are needed. The data storage efficiency was measured as the ratio of the size of stored files and the size of data which is stored for each file. The record size in our database was about 35 bytes, which cannot be compared to the stored data quantity. We may conclude that the data storage efficiency really only depends on the error correcting level used.

10.10 Conclusions

In this chapter we presented a solution for a cheap, reliable, high performance LAN-based distributed storage system. The basic idea is not new, but we could not find a system that had been optimized for such circumstances. The measurement results demonstrate the usability of this solution even with current desktop computing capabilities. We think that in the near future, with an increasing processor capacity, similar solutions will be widely used. The results shown in this thesis group except the churn estimation are all the results of the author and were published in research paper in research papers [Bil05] and [BD06].

Theses:

Thesis 12 *The Reed Solomon encoding-based error correction method is feassible soulution for file encoding with today's CPU power and the churn level we measured in the laboratory.*

Thesis 13 *The Paxos implementation with the optimized handling of non-requested messages is stable in a laboratory environment with the measured churn rate.*

Part III

Conclusions

11 Overview

The chief goal of this work was to highlight the importance of infrastructure awareness in the area of application development. The dissertation is based on two parts, and 5 thesis groups containing 13 theses.

In the first part we showed that while the network is moving in the direction of the context-based treatment of network flows, it is not well known that the number of flows provides serious scalability issues. We demonstrated that even with the currently available simple stateful services and high-end HW architectures there is a serious scalability weakness. In parallel with this we said that the number of flows produced by some P2P applications provide a significant fraction of the total number of parallel flows produced by end-users. Despite this fact, in the today's ISP friendly P2P systems concentrate only on the volume of the traffic and not on the composition of the traffic. These results were presented as two theses (Thesis 1, Thesis 2).

In the second part we provided four examples of infrastructure aware applications in four separate studies. The detectability of the P2P botnets is a serious question as they are becoming even more sophisticated. We showed that with a low-degree DHT-based approach one can construct botnets which are not detectable from a single point of the Internet. The results of this part are presented in three theses (Thesis 3, Thesis 4, Thesis 5).

Next, we studied the issue of low-degree DHT construction and we showed that it is possible to build scalable, robust low degree DHTs with the help of link grouping and gossip-based information exchange. The results of this part are presented in three theses (Thesis: 6, Thesis: 7, Thesis: 8).

SIP compression is not related to the P2P overlay and the number of flows directly, but it is related to the 3G links and the capabilities of the handheld devices. We showed that the classical compression algorithms are not suitable for the SIP compression task. We modified several well-known algorithms and found that some of these algorithms achieve the best compression ratios, while others are better from the point of view decoder delay. The results of this part are given in three theses (Thesis 9, Thesis 10, Thesis 11).

Another example of infrastructure awareness is the distributed storage system optimized for local communication and churn measured in the laboratories. Here we applied the classical Paxos algorithm with simplifications in order to achieve consistency even in the case of a constantly changing set of nodes. We showed that with Reed Solomon error correction and the modified Paxos algorithm one can build a reliable high performance local storage. The results of this part are given in two theses (Thesis 12, Thesis 13).

Here in thesis groups we summarize the publications:

	Contribution - short title	Theses	Publications
I/1	Flows vs. stateful services	Thesis 1, Thesis 2	[Bil06]
II/1	Hiding botnets	Thesis 3, Thesis 4, Thesis 5	[JB09b]
II/2	Low degree DHT with large churn	Thesis 6, Thesis 7, Thesis 7	[JB09a]
II/3	SIP compression	Thesis 9, Thesis 10, Thesis 11	[FBSS03]
II/4	Scalable storage	Thesis 12, Thesis 13	[Bil05] [BD06]

Table 11.1: Thesis contributions and supporting publications

Bibliography

[1] V. Aggarwal, O. Akonjang, and A. Feldmann. Improving user and isp experience through isp-aided p2p locality. In *Proceedings of IEEE Global Internet Symposium*, 2008.

[2] V. Aggarwal, A. Feldmann, and C. Scheideler. Can ISPS and P2P users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review*, 37(3):40, 2007.

[3] O. Akonjang, A. Feldmann, S. Previdi, B. Davie, and D. Saucez. The PROXIDOR service. *Internet Engineering Task Force, Internet-Draft draft-akonjang-alto-proxidor-00, Mar*, 2009.

[4] Antonio Pescapé Alessio Botta, Alberto Dainotti. Multi-protocol and multi-platform traffic generation and measurement. In *INFOCOM 2007 DEMO Session*. INFOCOM, 2007. `http://www.grid.unina.it/software/ITG/documentation.php`.

[5] J. D. Kubiatowicz B. Y. Zhao and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant widearea location and routing. *Symposium on Operating Systems Design and Implementation*, Tech. Rep. UCB/CSD-01-1141, 2001.

[6] J. Bannister, P.M. Mather, and S. Coope. *Convergence technologies for 3G networks: IP, UMTS, EGPRS and ATM*. John Wiley & Sons Inc, 2004.

[7] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt. A comparative analysis of web and peer-to-peer traffic. In *Proceeding of the 17th international conference on World Wide Web*, pages 287–296. ACM, 2008.

[8] Guy E. Blelloch. *Introduction to Data Compression*. `www-2.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/compression.pdf`.

[9] M. Castro and B. Liskov. Proactive recovery in a byzantine-fault-tolerant system. *In Proc. of OSDI*, 2000.

[10] Senthilkumar G. Cheetancheri, John Mark Agosta, Denver H. Dash, Karl N. Levitt, Jeff Rowe, and Eve M. Schooler. A distributed host-based worm detection system. In *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense (LSAD'06)*, pages 107–113, New York, NY, USA, 2006. ACM.

[11] Michael Ching. Packet buffer memory bandwidth causes npu performance bottlenecks. 2003.

[12] D.R. Choffnes and F.E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. *ACM SIGCOMM Computer Communication Review*, 38(4):363–374, 2008.

[13] Cisco. Cisco aon: A network embedded intelligent message routing system. Product documentation, Cisco Systems, 2005.

[14] Cisco. Cisco application extension platform overview. Product documentation, Cisco Systems, 2008.

[15] Cisco. Cisco visual networking index: Forecast and methodology, 2008-2013. Product documentation, Cisco Systems, 2010.

[16] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2009-2014. Product documentation, Cisco Systems, 2010.

[17] B. Claise. Cisco systems netflow services export version 9, 2004.

[18] M. D. Cookson and D. G. Smith. 3G service control. *British Telecom Technology Journal*, 19, 2001. www.sipcenter.com/files/3G_Service_Ctl.pdf.

[19] Colin Cooper and Alan Frieze. Hamilton cycles in random graphs and directed graphs. *Random Structures and Algorithms*, 16(4):369–401, 2000.

[20] Ubiquity Software Corporation. *SIP Enhanced Mobile Network*. `www.sipcenter.com/aboutsip/sipmobil.htm`.

[21] V. Bilicki Cs. Imreh. On the optimization models of congestion control. *In XXVI. Operational Research Conference*, 2004.

[22] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26. ACM, 2004.

[23] David Dagon. Botnet detection and response: The network is the infection, 2005. OARC Workshop presentation.

[24] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification [RFC 2460]*. `www.ietf.org/rfc/rfc2460.txt`, Dec 1998.

[25] P. Deutch. *DEFLATE Compressed Data Format Specification version 1.3 [RFC 1951]*. `http://www.ietf.org/rfc/rfc1951.txt`, May 1996.

[26] J. R. Douceur and W. J. Bolosky. A large-scale study of filesystem contents. *In ACM SIGMETRICS'99*, pages 59–71, 1999.

[27] K. Dubray. Terminology for ip multicast benchmarking[rfc2432]. Technical report, IETF, 1998. `http://www.faqs.org/rfcs/rfc2432.html`.

[28] N. Duffield and C. Lund. Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, page 191. ACM, 2003.

[29] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better NetFlow. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 245–256. ACM, 2004.

[30] Bill Fenner, Mark Handley, Hugh Holbrook, and Isidor Kouvelas. Protocol independent multicast - sparse mode (pim-sm):protocol specification (revised). Technical report, IETF, 2006. `http://tools.ietf.org/wg/pim/draft-ietf-pim-sm-v2-new/draft-ietf-pim-sm-v2-new-12.txt`.

[31] M. Gracia, C. Bromann, J. Ott, R. Price, and A. Roach. *The Session Initiation Protocol (SIP) and Session Description Protocol (SDP) static dictionary for Signaling Compression (SigComp) [Internet Draft]*. `www.ietf.org/internet-drafts/draft-ietf-sipping-sigcomp-sip-dictionary-03.txt`, 2002.

[32] J.B. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[33] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[34] H. Haddadi, R. Landa, A.W. Moore, S. Bhatti, M. Rio, and X. Che. Revisiting the issues on netflow sample and export performance. In *Communications and Networking in China, 2008. ChinaCom 2008. Third International Conference on*, pages 442–446, 2008.

[35] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. *SIP: Session Initiation Protocol [RFC 2543]*. `www.ietf.org/rfc/rfc2543.txt`, March 1999.

[36] Hans Hannu. *Signaling Compression Requirements & Assumptions*. `http:www.ietf.org/internet-drafts/draft-ietf-rohc-signaling-req-assump-06.txt`, Jun 2002.

[37] J. H. Hartman and J. K. Ousterhout. The zebra striped network file system. *Operating Systems Review - 14th ACM Symposium on Operating System Principles*, 25(1):29–43, 1993.

[38] M. Hidell. Decentralized modular router architectures. 2006.

[39] Erik Hjelmvik. The SPID Algorithm. 2008.

[40] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, Berkeley, CA, USA, 2008. USENIX Association.

[41] T. Hoßfeld, D. Hausheer, F. Hecht, F. Lehrieder, S. Oechsner, I. Papafili, P. Racz, S. Soursos, D. Staehle, G.D. Stamoulis, et al. An economic traffic management approach to enable the triplewin for users, ISPs, and overlay providers. *Towards the Future Internet*, page 24, 2009.

[42] Paul G. Howard and Jeffrey S. Vitter. Practical Implementations of Arithmetic Coding. *Image and Text Compression*, 1992.

[43] Paul G. Howard and Jeffrey S. Vitter. Fast progressive lossless image compression. 1994.

[44] Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems*. PhD thesis, citeseer.nj.nec.com/howard93design.html, 1993.

[45] C.H. Hsu and M. Hefeeda. ISP-friendly peer matching without ISP collaboration. In *Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–6. ACM, 2008.

[46] C. Huitema. *The new Internet Protocol*. Prentice Hall, 1996.

[47] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, k claffy, and Colleen Shannon. The IPv4 Routed /24 AS Links Dataset – 2008-01-02. http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml.

[48] IETF. *ROHC group*. http:www.ietf.org/html.charters/rohc-charter.html.

[49] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC'07)*, pages 315–320, New York, NY, USA, 2007. ACM.

[50] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, George Varghese, and Hyunchul Kim. Graption: Automated detection of P2P applications using traffic dispersion graphs (TDGs). Technical Report UCR-CS-2008-06080, Department of Computer Science and Engineering, University of California, Riverside, June 2008.

[51] Intel. *Intel PXA800F Cellular Processor Developer Manual*. http://www.intel.com/design/pca/prodbref/252336.htm?iid=devnav_btn1+hw_proc_wap_pxa800f&.

[52] ITU. Series z: Languages and general software aspects for telecommunication systems formal description techniques (fdt) message sequence chart. Technical report, ITU-T, 1999. http://www.itu.int/ITU-T/studygroups/com10/languages/Z.120_1199.pdf.

[53] M. L. Scheinholtz C. A. N. Soules J. D. Strunk, G. R. Goodson and G. R. Ganger. Self-securing storage: protecting data in compromised systems. *Symposium on Operating Systems Design and Implementation*, pages 165–180, 2000.

[54] S. G. Menees D. A. Nichols M. Satyanarayanan R. N. Sidebotham J. H. Howard, M. L. Kazar and M. J. West. Scale and performance in a distributed file system. *Transactions on Computer Systems*, 6:51 – 81, 1988.

[55] T. Bates J. Hawkinson. Guidelines for creation, selection, and registration of an autonomous system (as). IETF-RFC, IETF, 1996.

[56] G. R. Ganger J. J. Wylie, G. R. Goodson and M. K. Reiter. Efficient byzantine-tolerant erasure-coded storage. *In Int. Conf. on Dependable Systems and Networks (DSN)*, 2004.

[57] J. Strunk G. Ganger H. Kiliccote P. Khosla J. Wylie, M. Bigrigg. Survivable information storage systems. *IEEE Computer*, 33(8):61–68, 2000.

[58] Van Jacobson. *Compressing TCP/IP headers for Low-Speed Serial links [RFC 1144]*. http://www.ietf.org/rfc/rfc1144.txt, Feb 1990.

[59] Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based overlay topology management. In Sven A. Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors, *Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2006.

[60] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, 2009.

[61] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):8, August 2007.

[62] C. Jennings and D.A. Bryan. P2P For Communications: Beyond File Sharing. *Business Communications Review*, 36(2):36, 2006.

[63] David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, 1997.

[64] P. Francis K. Egevang. The ip network address translator (nat), 1994.

[65] Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, and Stefan Savage. The heisenbot uncertainty problem: Challenges in separating bots from chaff. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, Berkeley, CA, USA, 2008. USENIX Association.

[66] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probablistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), March 2003.

[67] Anne-Marie Kermarrec and Maarten van Steen, editors. *ACM SIGOPS Operating Systems Review 41*. October 2007. Special issue on Gossip-Based Networking.

[68] S. Kiesel and M. Stiemerling. Alto h12, 2009.

[69] Freddy Kharoliwalla Kiran Mukesh Misra. Study of internet router architectures. 2001.

[70] J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[71] J. Kleinberg. The wireless epidemic. *Nature*, 449:287–288, 2007. News and Views.

[72] Jon Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

[73] Jon Kleinberg. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC'00)*, pages 163–170, New York, NY, USA, 2000. ACM.

[74] T. Kocak and F. Basci. A power-efficient TCAM architecture for network forwarding tables. *Journal of Systems Architecture*, 52(5):307–314, 2006.

[75] Joseph S. Kong, Jesse S. A. Bridgewater, and Vwani P. Roychowdhury. A general framework for scalability and performance analysis of DHT routing systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 343–354, Washington, DC, USA, 2006. IEEE Computer Society.

[76] R. Gopal L. Yang, T. Anderson. Forwarding and control element separation (forces) framework. IETF-RFC, IETF, 2003.

[77] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[78] L. Lamport. Paxos made simple. *ACM SIGACT News Distributed Computing Column*, 32(4), 2001.

[79] Edward K. Lee and Chandrohan A. Thekkah. Petal: distributed virtual discs. *SIGPLAN Notices*, 31(9):84–92, 1996.

[80] J. Lilley, J. Yang, H. Balakrishan, and S. Seshan. *A Unified Header Compression Framework for Low Bandwith Links*. MIT Laboratory for Computer Science.

[81] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(2):72–93, 2005.

[82] M. Taqqu M. Crovella and A. Bestavros. Heavy-tailed probability distributions in the world wide web. *A Practical Guide To Heavy Tails: Statistical Techniques and Applications*, 1998.

[83] Priya Mahadevan, Dmitri Krioukov, Marina Fomenkov, Xenofontas Dimitropoulos, k c claffy, and Amin Vahdat. The Internet AS-level topology: three data sources and one definitive metric. *SIGCOMM Comput. Commun. Rev.*, 36(1):17–26, 2006.

[84] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2002.

[85] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003.

[86] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04)*, pages 54–63, New York, NY, USA, 2004. ACM.

[87] E. Marocco and V. Gurbani. Application-Layer Traffic Optimization (ALTO) Problem Statement. *ID, draft-marocco-alto-problem-statement-03 (Work in Progress)*, 2009.

[88] Laurent Massoulié, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS 2003)*, pages 47–55, Florence, Italy, 2003.

[89] John P. John Arvind Krishnamurthy Michael Piatek, Harsha V. Madhyastha and Thomas Anderson. Pitfalls for ISP-friendly P2P design. *In Proc. of HotNets-VIII.*, 2009.

[90] Alberto Montresor. A robust protocol for building superpeer overlay topologies. In *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing (P2P'04)*, pages 202–209, Zurich, Switzerland, August 2004. IEEE Computer Society.

[91] Moni Naor and Udi Wieder. Know thy neighbor's neighbor: Better routing for skip-graphs and small worlds. In *Peer-to-Peer Systems III*, volume 3279 of *Lecture Notes in Computer Science*, pages 269–277. Springer, 2005.

[92] Nortel Networks. *A Comparison Between GERAN Packet-Switched Call Setup Using SIP and GSM Circuit-Switched Call Setup Using RIL3-CC, RIL3-MM, RIL3-RR, and DTAPRev. 0.3*, October 2000.

[93] Thuy T. T. Nguyen and Grenville Armitage. A survey of techniques for Internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 10(4):56–76, 2008.

[94] G. A. Gibson R. H. Katz D. A. Patterson P. M. Chen, E. K. Lee. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys*, 1994.

[95] Michael Patterson. NetFlow overflow with TCAM tables. http://www.lovemytool.com/blog/2009/12/netflow-overflow-with-tcam-tables-by-michael-patterson.html.

[96] PeerSim. http://peersim.sourceforge.net/.

[97] C. Popoviciu, A. Hamza andG. Van de Velde, and D. Dugatkin. Ipv6 benchmarking methodology. Technical report, IETF, 2006. `http://potaroo.net/ietf/idref/draft-popoviciu-bmwg-ipv6benchmarking/`.

[98] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran. A foray into Conficker's logic and rendezvous points. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*. USENIX, 2009.

[99] J. Postel. *Internet Protocol [RFC 0791]*. `www.ietf.org/rfc/rfc0791.txt`, Sept 1981.

[100] R. Price, H. Hannu, C. Bormann, J. Christoffersson, Z. Liu, and J. Rosenberg. *Signaling Compression [Internet Draft]*. `www.ietf.org/internet-drafts/draft-ietf-rohc-sigcomp-04.txt`, Feb 2002.

[101] R. Price, H. Hannu, C. Bormann, J. Christoffersson, Z. Liu, and J. Rosenberg. *Signaling Compression [Internet Draft]*. `www.ietf.org/internet-drafts/draft-ietf-rohc-sigcomp-07.txt`, June 2002.

[102] R. Price, H. Hannu, C. Bormann, J. Christoffersson, Z. Liu, and J. Rosenberg. *Signaling Compression Reference [Internet Draft]*. `www.ietf.org/internet-drafts/draft-ietf-rohc-sigcomp-05.txt`, Mar 2002.

[103] R. Price, J. Rosenberg, C. Bormann, H. Hannu, and Z. Liu. *Universal Decompressor Virtual Machine (UDVM) [Internet Draft]*. `www.ietf.org/internet-drafts/draft-ietf-rohc-sigcomp-udvm-00.txt`, Jan 2002.

[104] Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'06)*, 2006.

[105] Khalid Raza. *Large-scale IP network solutions*. Macmillan, Indianapolis IN, 2000.

[106] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[107] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27:24–36, 1997.

[108] Xiaoyuan Yang Georgos Siganos Pablo Rodriguez Ruben Cuevas, Nikolaos Laoutaris. Deep diving into BitTorrent locality. 2009.

[109] Y. Saito S. Spence S. Frolund, A. Merchant and A. Veitch. Fab: Enterprise storage systems on a shoestring. *In 8th Workshop on Hot Topics in Operating Systems (HOTOSVIII)*, 2003.

[110] D. Geels H. Weatherspoon B. Zhao S. Rhea, P. Eaton and J. Kubiatowicz. Pond: The oceanstore prototype. *In Proceedings of the Conference on File and Storage Technologies (FAST)*, 2003.

[111] Hugo Santos. Mrd6, an ipv6 multicast router. Technical report, 2006. `http://fivebits.net/proj/mrd6/`.

[112] Hugo Santos. dbeacon, a multicast beacon, 2007. `http://fivebits.net/proj/dbeacon/`.

[113] J. Seedorf, S. Kiesel, and M. Stiemerling. Traffic Localization for P2P-Applications: The ALTO Approach. 2009.

[114] L. Sógor, M. Fidrich, L. Martonossy, P. Hendlein, and G. Somlai. Comparison of inter-operation mechanisms between IPv4 and IPv6. In *9th IFIP working Conference on Performance Modelling and Evaluation of ATM & IP networks*, June 2001.

[115] L. Sógor, P. Hendlein, K. Notaisz, M. Fidrich, G. Fóris, and G. Kiss. Development of a Communication Environment between IPv6 and IPv4. In *7th Int. Symposium on Programming Languages and Software Tools (SPLST)*, June 2001. An extended version is accepted for publication in Acta Cybernetica.

[116] Zhijie Shen and Roger Zimmermann. Isp-friendly peer selection in p2p networks. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 869–872, New York, NY, USA, 2009. ACM.

[117] M. Steiner and E. Biersack. Where Is My Peer? Evaluation of the Vivaldi Network Coordinate System in Azureus. *NETWORKING 2009*, pages 145–156, 2009.

[118] Henry Stern. Effective malware: The trade-off between size and stealth. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*. USENIX, 2009. invited talk.

[119] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, San Diego, CA, 2001. ACM, ACM Press.

[120] D. Stopp and B. Hickman. Methodology for ip multicast benchmarking[rfc3918]. Technical report, IETF, 2004. `http://www.faqs.org/rfcs/rfc3918.html`.

[121] W. Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In Wenke Lee, Cliff Wang, and David Dagon, editors, *Botnet Detection: Countering the Largest Security Threat*, volume 36 of *Advances in Information Security*, pages 1–24. Springer, 2008.

[122] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC'06)*, pages 189–202, New York, NY, USA, 2006. ACM.

[123] Cisco Systems. Netflow performance analysis. White Paper, Cisco Sytems, 2007.

[124] Cisco Systems. Netflow services solutions guide. White Paper, Cisco Sytems, 2007.

[125] Cisco Systems. Network management case study: How cisco it uses netflow to capture network behavior, security, and capacity data. White Paper, Cisco Sytems, 2007.

[126] Cisco Systems. Network based application recognition performance analysis. White Paper, Cisco Sytems, 2008.

[127] Cisco Systems. Network based application recognition. White Paper, Cisco Sytems, 2010.

[128] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81–94, 2007.

[129] G. Trotter. Terminology for forwarding information base (fib) based router performance, 2001.

[130] T. Li V. Fuller. Classless inter-domain routing (cidr): The internet address assignment and aggregation plan. IETF-RFC, IETF, 2006.

[131] A. Vemuri and J. Peterson. *SIP for Telephones (SIP-T): Context and Architectures [Internet Draft]*. `www.softarmor.com/sipping/drafts/rfced/draft-ietf-sipping-sipt-04.txt`, Jun 2002.

[132] R. Vida and L. Costa. Multicast listener discovery version 2 (mldv2) for ipv6. Technical report, IETF, 2004. `http://www.ietf.org/rfc/rfc3810.txt`.

[133] N. Weaver, D. Ellis, S. Staniford, and V. Paxson. Worms vs. perimeters: the case for hard-LANs. In *Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects (HOTI'04)*, pages 70–76, Washington, DC, USA, 2004. IEEE Computer Society.

[134] H. Xie, Y.R. Yang, A. Krishnamurthy, Y.G. Liu, and A. Silberschatz. P4P: Provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362, 2008.

[135] H. Zang and A. Nucci. Traffic monitor deployment in IP networks. *Computer Networks*, 53(14):2491–2501, 2009.

References

[BD06] Vilmos Bilicki and József Dombi. Building a general framework for the consistency management of distributed applications. In *.NET Technologies 2006 conference proceedings(.NET'06)*, volume 3, pages 55–63, Plzen, Czeck Republic, 2006. University of West Bohemia. http://dotnet.zcu.cz/NET_2006/Papers_2006/!Proceedings_Full_Papers_2006.pdf.

[Bil05] Vilmos Bilicki. Lanstore: a highly distributed reliable file storage system. In *.NET Technologies2005 conference proceedings(.NET'05)*, volume 3, pages 47–57, Plzen, Czeck Republic, 2005. University of West Bohemia. http://wscg.zcu.cz/ROTOR/Journal/Archive/2005_Vol_3.pdf#page=59.

[Bil06] Vilmos Bilicki. Testing and verifying an ipv6 based multicast network. In *ICCGI '06: Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, pages 3–13, Washington, DC, USA, 2006. IEEE Computer Society.

[FBSS03] Manta Fidrich, Vilmos Bilicki, Zoltan Sogor, and Gabor Sey. Sip compression. *Periodica Polytechnica, Electrical Engineering.*, 47(1-2):37–56, 2003. http://www.inf.u-szeged.hu/~bilickiv/research/CSCS2002.ps.

[JB09a] Márk Jelasity and Vilmos Bilicki. Scalable p2p overlays of very small constant degree: An emerging security threat. In *SSS '09: Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 399–412, Berlin, Heidelberg, 2009. Springer-Verlag.

[JB09b] Márk Jelasity and Vilmos Bilicki. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*, pages 1–8. USENIX, 2009. http://www.usenix.org/events/leet09/tech/.

Appendices

A
Summary

The chief goal of this work was to highlight the importance of infrastructure awareness in the area of application development. The dissertation is based on two parts, and 5 thesis groups containing 13 theses.

In the first part we showed that while the network is moving in the direction of the context-based treatment of network flows, it is not well known that the number of flows provides serious scalability issues. We demonstrated that even with the currently available simple stateful services and high-end HW architectures there is a serious scalability weakness. In parallel with this we said that the number of flows produced by some P2P applications provide a significant fraction of the total number of parallel flows produced by end-users. Despite this fact, in the today's ISP friendly P2P systems concentrate only on the volume of the traffic and not on the composition of the traffic. These results were presented as two theses:

- Thesis 1: The performance of stateful services in the distribution and core layers depends heavily on the number of unicast or multicast flows.
- Thesis 2: The ISP friendly P2P system must take into account the number of flows generated by the overlay too.

The results shown in this thesis group are all the results of the author. The results related to the multicast traffic of this contribution point were published in research paper [Bil06].

In the second part we provided four examples of infrastructure aware applications in four separate studies. The detectability of the P2P botnets is a serious question as they

are becoming even more sophisticated. We showed that with a low-degree DHT-based approach one can construct botnets which are not detectable from a single point of the Internet. The results of this part are presented in three theses:

- Thesis 3: It is possible to build P2P botnet which could not be detected with the help of the TDG method from a single point (AS).
- Thesis 4: With the help of localization and link clustering the P2P botnet can hide from a single point TDG-based monitoring.
- Thesis 5: Localization has a minor effect on the hiding capability of the P2P botnet, while link clusterization has significant effects on the visibility of the P2P botnet.

The results of this contribution point were published in research paper [JB09b]. The results connected to the Thesis 5 are the results of the author. The rest are the results of shared work.

Next, we studied the issue of low-degree DHT construction and we showed that it is possible to build scalable, robust low degree DHTs with the help of link grouping and gossip-based information exchange. The results of this part are presented in three theses:

- Thesis 6:The Symphony topology is scalable; that is, $\lim_{h\to\infty} p(h,q) > 0$, if (i) all the links have $O(\log N)$ backup links, or if (ii) all the links have $O(\log\log N)$ backup links and $q \leq e^{-2} \approx 0.135$.
- Thesis 7: It is possible to build a low degree DHT that is robust even in the case of large churn.
- Thesis 8: The T-Man based DHT exetended with the four methods (Lookahead, Degree balancing, Stratification and Short-link avoidance) is stable in the case of significant churn (Weibull k=0.5)

The results of this contribution point were published in research papers [JB09a]. The results connected to Thesis 8 are the results of the author. The rest are the results of shared work.

SIP compression is not related to the P2P overlay and the number of flows directly, but it is related to the 3G links and the capabilities of the handheld devices. We showed that the classical compression algorithms are not suitable for the SIP compression task. We modified several well-known algorithms and found that some of these algorithms achieve the best compression ratios, while others are better from the point of view decoder delay. The results of this part are given in three theses:

- Thesis 9:The classical compression algorithms without modification are not applicable for SIP compression.

- Thesis 10:The modified Deflate (run length encoding + context modeling) algorithm is the best SIP when the call setup delay is important.
- Thesis 11:The modified LZ77 algorithm is the fastest on the decompressor side. Context modelling (i.e. prefix-free encoders) is the fastest on the compressor side.

The results shown in this thesis group are all the results of the author and were published in research paper [FBSS03].

Another example of infrastructure awareness is the distributed storage system optimized for local communication and churn measured in the laboratories. Here we applied the classical Paxos algorithm with simplifications in order to achieve consistency even in the case of a constantly changing set of nodes. We showed that with Reed Solomon error correction and the modified Paxos algorithm one can build a reliable high performance local storage. The results of this part are given in two theses:

- Thesis 12:The Reed Solomon encoding-based error correction method is feassible soulution for file encoding with today's CPU power and the churn level we measured in the laboratory.
- Thesis 13: The Paxos implementation with the optimized handling of non-requested messages is stable in a laboratory environment with the measured churn rate.

The results shown in this thesis group except the churn estimation are all the results of the author and were published in research paper in research papers [Bil05] and [BD06].

Here in thesis groups we summarize the publications:

	Contribution - short title	Theses	Publications
I/1	Flows vs. stateful services	Thesis 1, Thesis 2	[Bil06]
II/1	Hiding botnets	Thesis 3, Thesis 4, Thesis 5	[JB09b]
II/2	Low degree DHT with large churn	Thesis 6, Thesis 7, Thesis 7	[JB09a]
II/3	SIP compression	Thesis 9, Thesis 10, Thesis 11	[FBSS03]
II/4	Scalable storage	Thesis 12, Thesis 13	[Bil05] [BD06]

Table A.1: Thesis contributions and supporting publications

B
Összefoglaló

Az értekezés célja, hogy felvázolja, a legfontosabb trendeket melyek meghatározzák a hálózatok fejlődési irányait és rávilágítson néhány területre ahol szükséges illetve célszerű az alkalmazásoknak alkalmazkodni a hálózat képességeihez. Az értekezés két nagy részre ezen belül öt tézis csoportra illetve tizenhárom tézisre bontva tárgyalja az alkalmazások és a hálózat viszonyait.

Az első részben az IP hálózat és ezen belül az Internet lehetséges fejlődési irányait vázoltuk fel. Itt bemutattuk azt, hogy egy fontos trend lehet a kontextus alapú kiszolgálás biztosító szolgáltatások megjelenése. Ezen szolgáltatások sokkal komolyabb erőforrás igénnyel fognak rendelkezni, mint a napjainkban is megtalálható állapottartó működést igénylő megoldások. Megvizsgáltuk, hogy az állapotok kezelése milyen nehézségekbe ütközik a különböző hálózati rétegeket kiszolgáló aktív eszközökön és mérésekkel prezentáltuk a mai eszközök teljesítménybéli korlátait. Megállapítottuk, hogy az aktív eszközökön kezelt kapcsolatok száma jelenti az állapottartó szolgáltatások kritikus pontját. Ezután bemutattuk azt, hogy a P2P alkalmazások egy része igen sok kapcsolatot tart fenn ami a fentiek fényében a kiszolgáló hálózaton skálázhatósági problémákat fog felvetni, illetve vet fel már ma is. A megállapításainkat az alábbi tézisekbe foglaltuk össze:

- 1. Tézis: Az állapottartó szolgáltatások teljesítménye a különböző hálózati rétegekben nagymértékben függ a kezelt pont-pont és pont-több pont típusú kapcsolatok számától.

- 2. Tézis: Az ISP barát P2P alkalmazások fejlesztőinek figyelembe kell venniük az alkalmazás által generált kapcsolatok számát is.

A tézis csoportban bemutatott eredmények a szerző eredményei. A csoportküldés teljesítményével kapcsolatos eredmények a [Bil06] cikkben lettek publikálva.

A második részben négy példát mutattunk be az infrastruktúrához alkalmazkodó alkalmazásokra. A P2P botnet-ek detektálása egyre több fejtörést okoz a hálózatok működtetőinek és a kutató közösségnek is. Rámutattunk arra, hogy kis fokszámú DHT alapokon készíthető olyan botnet ami TDG módszerrel gyakorlatilag detektálhatatlan az Internet tetszőlegesen választott pontjából. Ebből az eredményből az következik, hogy a különböző hálózat üzemeltetőknek együtt kell működniük a botnet-ek felderítésében. Az eredményeket három tézisen fogalmaztuk meg:

- 3. Tézis: Lehetséges olyan P2P botnet-et készíteni, amely nem detektálható TDG módszerrel egyetlen tetszőlegesen kiválasztott autonóm rendszerből sem.
- 4. Tézis: Lokalizáció és kapcsolat klaszterezés segítségével elérhető, hogy a botnet felderíthetetlen lesz a TDG módszerrel amennyiben csak egy autonóm rendszerből vizsgáljuk.
- 5. Tézis: A kapcsolatok klaszterezése jelentős hatással bír a botnet rejtőzködésére míg a lokalizáció hatása kevésbé jelentős.

A téziscsoportban megfogalmazott eredemények a [JB09b] cikkben voltak publikálva. Az 5. Tézis a szerző saját eredménye, míg a többi a társszerzővel elért közös eredmény.

Ezek után azt vizsgáltuk meg, hogy lehetséges-e olyan az előző téziscsoportokban fontosnak tartott kis fokszámú DHT alapú P2P algoritmus létrehozása, amely skálázható és robosztus is. Elméleti és kísérleti eredmények segítségével bemutattuk, hogy lehetséges amennyiben néhány általunk javasolt eljárást alkalmazunk. Az eredmények tézisként megfogalmazva:

- 6. Tézis: A Symphony alapú topológia skálázható azaz, $\lim_{h\to\infty} p(h,q) > 0$, amennyiben (i) minden kapcsolathoz van $O(\log N)$ tartalék kapcsolat, vagy amennyiben (ii) minden kapcsolatoz van $O(\log\log N)$ tartalék kapcsolat és $q \leq e^{-2} \approx 0.135$.
- 7. Tézis: Lehetséges hatékony és nagyon változékony hálózaton is üzembiztos kicsi fokszámú DHT alapú P2P algoritmust készíteni.
- 8. Tézis: A T-Man alapú pletyka hálózatot kiegészítve az alábbi algoritmusokkal: *Előretekintés, Fokszám biztosítás, Rétegezés, Rövid kapcsolatok elkerülése* a megvalósított DHT képes hatékonyan működni, és igen változékony hálózat (Weibull k=0.5) esetén is megőrzi stabilitását.

A téziscsoporthoz tartozó eredmények a [JB09a] cikkben lettek publikálva. A tézisek közül a 8. a szerző saját eredménye, míg a többi közös munka gyümölcse.

A SIP tömörítés ugyan nem kapcsolódik szorosan a P2P fedőhálózatokhoz és a kapcsolatok számához, de egy másik fontos területhez a 3G alapú rendszerekben használt protokolloknak viszont szerves része. Ebben a téziscsoportban azt elemeztük, hogyan lehet/érdemes a vezetékes közegre optimalizált SIP protokollt a vezetékmentes közegre alakítani. Kiderült, hogy változtatás nélkül nem érdemes a hagyományos tömörítési algoritmusokat alkalmazni. A 3G kapcsolatok felépítésénél nem csak a tömörítés, hanem a késleltetés, a processzor valamint a memória igény is fontos szempont. Miután megfelelően módosítottunk néhány jól ismert tömörítési algoritmust kiderült, hogy érdemes őket használni viszont nincs optimális megoldás. A legfontosabb megállapításainkat az alábbi tézisekben fogalmaztuk meg:

- 9. Tézis: Változtatás nélkül a klasszikus tömörítési algoritmusokat nem érdemes SIP tömörítésre használni.
- 10. Tézis: A módosított Deflate algoritmus (futáshossz kódolás + kontextus modellezés) teljesít legjobban amennyiben a SIP kapcsolatfelépítés időtartama a döntő.
- 11. Tézis: A módosított LZ77-es algoritmus a leggyorsabb a kitömörítő oldalon míg a tömörítésnél a kontextus modellezésen alapuló eljárások a leggyorsabbak (pl.: előtag mentes eljárások).

A téziscsoportban leírt eredmények a [FBSS03] cikkben voltak publikálva és az elérésükben szerző munkája volt a meghatározó.

Egy másik érdekes példa a hálózati környezethez alkalmazkodásra egy olyan elosztott fájl tároló rendszer amely a számítógépes laboratóriumokra jellemző változékonyságban is képes megbízhatóan és hatékonyan működni. A konzisztencia biztosításához a klasszikus Paxos algoritmust és annak néhány általunk javasolt kiegészítését használtuk fel. Az adatok redundanciáját a szintén klasszikus Reed Solomon algoritmussal biztosítottuk. A téziscsoportban bemutattuk, hogy a fent említett algoritmusokra alapozva lehetséges nagy teljesítményű, skálázható és robosztus elosztott tárolót építeni.

- 12. Tézis: A napjainkban rendelkezésre álló személyi számítógépek számítási kapacitása alkalmas arra, hogy a Reed Solomon alapú kódoláson alapuló hibajavító megoldásokat használjunk a laboratóriumi változékonyság okozta hibák elfedésére.
- 13. Tézis: Az általunk kiegészített Paxos alkalmas arra, hogy a számítógépes laboratóriumokban tapasztalt változékonyság esetén biztosítsa a konzisztenciát.

A téziscsoport eredményei az alábbi cikkekben lettek publikálva: [Bil05], [BD06]. Az itt megfogalmazott eredményeknél a laboratóriumi változékonyság mérését kivéve a szerző hozzájárulása volt a meghatározó.

Végezetül egy összesítés a téziscsoportokról és a megfelelő publikációkról:

	Tézis csoport	Tézisek	Publikáció
I/1	Kapcsolatok és állapottartó szolgáltatások	1. Tézis, 2. Tézis	[Bil06]
II/1	Rejtőző botnet-ek	3. Tézis,4. Tézis, 5. Tézis	[JB09b]
II/2	Kis fokszámú DHT változékony hálózatban	6. Tézis, 7. Tézis, 7. Tézis	[JB09a]
II/3	SIP tömörítés	9. Tézis, 10. Tézis, 11. Tézis	[FBSS03]
II/4	Skálázható tároló	12. Tézis, 13. Tézis	[Bil05] [BD06]

Table B.1: Tézis csoportok, tézisek és publikációk