Autonomous Systems and Inter-Domain Routing in the Internet

A study into the possibilities and usefulness of visualisation of the Internet topology at the level of Autonomous Systems

Autonomous Systems and Inter-Domain Routing in the Internet

A study into the possibilities and usefulness of visualisation of the Internet topology at the level of Autonomous Systems

Student :	Marnix Kaart
Supervisors :	Prof. Dr. W.G. Vree
	Ir. J.G. Snip
	Drs. J.P. van Best
	Drs. W.G. van den Berg
Date :	23 September 2001
Status :	Final concept

Preface

As part of the UMEEPI project led by prof. dr. W.G. Vree, from Delft University of Technology, a thesis study has been performed. The UMEEPI project (Understanding and Modelling of End-to-End Performance of the Internet) is a collaboration of Delft University of Technology, RIPE NCC, KPN Research and the University of Amsterdam.

The thesis study has been done at KPN Research in Leidschendam, under the daily supervision of Jan Gerard Snip. The supervisors from Delft University of Technology are drs. J.P. van Best, drs. Wander van den Berg and prof. dr. W.G. Vree.

Within the UMEEPI project data from the RIPETT project from RIPE-NCC has been used. This data has been collected in co-operation with different ISPs across Europe and America and therefore may not be used for any commercial goals.

This paper presents the results, conclusions and recommendations found within this thesis study.

Contents

1. Introduction

The research effort described here focuses on topology maps of the Internet at the Autonomous Systems (AS) level. This study tries to determine whether such maps can be derived and tries to give insight in the usability of such maps in understanding and modelling the end-to-end performance of the Internet. To this end a prototype of a visualisation tool has been developed and some topology maps have been made from the data.

This research is part of a broader research project: UMEEPI, meaning Understanding and Modelling of the End-to-End Performance of the Internet. The goal of this UMEEPI project is to understand and model IP packet dynamics on the Internet, with the main focus on identifying and understanding bottlenecks [VR00]. Within this project traceroute data from the RIPE Test Traffic project will be used, which will be described later in this chapter.

In this chapter, the UMEEPI project and the related RIPETT project are briefly described. At this time some terminology and concepts are left unexplained and might be difficult to understand. Still it was chosen to show the context of this thesis study before going into the details of the actual problem, which will be done in the second paragraph. There a problem analysis is made where terms and concepts are explained more thoroughly. Also the scope of the study is determined here. After this the problem is formulated by presenting the research goals and research questions. Finally the research design is described, stating how and by what means the research questions should be answered.

In Chapter 2 some of the concepts and terminology of the Internet are discussed to provide us with a common understanding and vocabulary. Also a brief history of the Internet is given. In Chapter 3 the concepts of routing, Autonomous Systems, the inter-domain routing protocol BGP4, policy routing and peering are described. Extensive insight herein has proven necessary in order to continue the study of topology maps at the AS level. Chapter 4 shows some initial exploration and analysis of the available (IP level) data from the RIPETT project and also an initial version of the prototype visualisation tool and some of its results are described here. The resulting visualisations of the topology are still at the IP level but in Chapter 5 the focus will shift to visualisation of the topology at the AS level. First a method is shown to map IP addresses onto AS numbers and the necessary enhancements of the visualisation tool are described. Finally this chapter contains some interesting results, which consist of different visualisations of the topology at the AS level. The final chapter, Chapter 6, contains the conclusions and recommendations that where found in this study.

The UMEEPI project

This thesis study is part of the UMEEPI project which uses the measurements from the RIPETT project and possibly those of the Surveyor project, to understand and model the packet dynamics on the Internet. The RIPETT project will be described later on. The UMEEPI project is a collaboration of the Delft University of Technology, the Amsterdam University, KPN Research and RIPE-NCC. A summary of the research proposal by prof. dr. W.G. Vree [VR00] will be presented here. For a complete understanding, however, the next paragraph, problem analysis, should also be read.

We want to understand the mechanisms behind the ever changing Internet. We clearly concentrate on the lower level packet dynamics in an attempt to model and simulate the mechanisms that underlie the measured behaviour. The main focus is on identifying and understanding bottlenecks and end-to-end performance. This approach can possibly be extended to other IP networks. Here we may use useful modelling concepts that are developed and measured in [PA94, PA97, PA99], like bottleneck bandwidth, loss "outrages", self-similar behaviour, collective behaviour, time compression and various asymmetries.

The research proposal is divided into two separate efforts:

a) To understand, model and simulate the packet dynamics as measured by RIPETT
b) To study the consequences of extending the approach of the IPPM metrics and the RIPETT measuring strategy into the domain of peripheral networks.

a) Modelling packet dynamics

We want to use the characteristics of the new data that the RIPE test boxes provide to understand and model the one-way-delay and -loss figures. In addition we are going to use the GPS positional data of the test boxes to relate performance to real physical distance, and the improved traceroute data [UKKW98, KZ99] to relate distance to network paths.

Physical interconnection of machines plays an important role in the study of distributed computing. Because the physical interconnection of the Internet is unknown up to a certain level of detail and because the interconnection graph also can not be assumed to be random, we would like to reconstruct the real interconnection graph from the experimental data. This graph will aid in our effort to identify and understand bottlenecks.

After deriving the physical structure of the graph a dynamic model for the changing physical connections can be made, which can be used to predict the flow of data through the network. The availability of data makes it possible to immediately test the accuracy of these predictions.

b) Extending the approach to peripheral networks

This part of the research proposal is about extending the IPPM metrics and the RIPETT measurement strategy into the domain of peripheral networks. This study may result in a new or modified design of the measurement- and analysis methods. We focus on consumer networks (network which connects customers to the ISP's backbone) and on industrial control networks. The information obtained in the study of consumer networks will be used to obtain a better end-to-end performance model. This subject will not be described here further since the main focus of this study lies with part a) of the UMEEPI project.

RIPETT

The Ripe Test-Traffic or RIPETT-project provides the data used in the UMEEPI project. Ripe has about 40 computers placed mainly in Europe, and some in America and New Zealand that actively generate test traffic and send this to one another. This gives us a large set of traceroute information and delay and loss figures. In this section a description of this RIPETT project will be given. See appendix F for a complete list of all testboxes.

The IP Performance Measurement working group [IPPM] of the IETF (Internet Engineering Task Force), has established well defined, neutral performance metrics with broad acceptance, inspired by Paxson's research efforts [PA94, PA97, PA99]. A framework for IP performance metrics by the IPPM can be found in [PAMM98]. Following these metrics RIPE-NCC started developing dedicated and accurate measurement boxes: RIPETT (Ripe Test Traffic Measurements) [UKKW98, HO98, OH97].

The purpose of the Test Traffic Measurements Project is to independently measure the performance (connectivity) parameters, such as delays and routing-vectors, between hosts on the Internet using test traffic boxes located in the networks of participating ISP's. The data collected is analysed for the purpose of network troubleshooting and capacity planning. It is not the purpose to compare different ISP's network performance nor should the data be used for marketing purposes of any kind. [RIPETT]. In addition to participation in the UMEEPI project, RIPE NCC is offering the test-boxes as a service to providers. The providers hosting a test-box can obtain some useful information from the data collected by these testboxes [RIPE37].

Involved parties

Along with the universities of Delft and Amsterdam, two other organisations are involved in the UMEEPI project. They're very briefly described here.

Ripe NCC

The RIPE Network Co-ordination Centre (RIPE NCC) is one of 3 Regional Internet Registries (RIR) which exist in the world today, providing allocation and registration services which support the operation of the Internet globally [RIPE].

A working group was formed to discuss the results of the Test Traffic Measurements Project and provide comment as to the future needs on the type of data and methods of collection and presentation. The scope of the working group is not limited to the RIPETT project, but it should allow any new performance measuring techniques to be included [RIPE-tt-wg].

KPN Research

KPN is very active in the field of turning the potential of new technology into concrete applications for its customers. The KPN Research Innovation Centre supplies KPN with new products, services and processes that make a substantial contribution towards realising the ambition to play a major European role in this field [KPN].

KPN has a lot of experience in doing research in the area of networks and their end to end quality assurance. Although initially directed towards the control and management of the transmission quality of public switched telephone networks, the inevitable evolution towards packet switching for all types of information services has stimulated new research in areas like routing, streaming, flow-control, new addressing schemes, etc., both from a theoretical and from a practical angle [VR00]. Telephony is becoming a mere member of a wide range of multimedia services, which makes the control and management of the end-to-end transport quality absolutely essential. There are still many unknowns in the timing and delays of the global Internet traffic.

We see that more and more services are TCP/IP and Internet oriented and this alone makes it important to have insight in the performance of the Internet. The theoretical skills and knowledge developed within this project can also be of great value to KPN. However, a lot of the networks of KPN are running ATM and this makes the relevance a little less obvious. But, still a lot of the above mentioned arguments for these kind of measurements and research herein hold for KPN.

Another reason for KPN to be involved in projects like these is that the Quality of Service of IP-based services is not only dependent on the own network, but also of the infrastructure and performance of other ISP’s networks. If we look at Internet Telephony, for example, the value of this service is probably also determined by the possibility for the clients of one network to be able to call clients of other networks. This inevitably means that the different networks operators are always dependent of the performance of other networks. It might well be possible to use Service Level Agreements to get some guarantee on the Quality, but without the means to measure the QoS and pinpoint bottlenecks, these SLAs will never be verifiable. Without research in these areas the deployment of new services will become more difficult.

Problem analysis

In this paragraph the problem is analysed in an effort to concretise the vague description of "lack of insight in end-to-end performance in the Internet" to a well defined and solvable problem formulation which is given in the next paragraph. Some concepts have already been mentioned in the previous paragraph.

Problem description

The Internet is and has been changing drastically in many respects, like the unexpected growth of the number of hosts, rapid paced improvement of technology and the transition of the organisation of large parts of the Internet from the traditional university community and the National Science Foundation (NSF) into the commercial domain, with its many global and local competitive Internet Service Providers (ISPs). These and other changes make it very hard to manage the network, to understand how it really functions and how the performance of the whole remains at an acceptable level and to predict its behaviour [VR00].

The transition into a competitive industry for Internet services has resulted in the fact that the global infrastructure of the Internet now consists of a complex array of telecommunications carriers and providers with an ever growing number of hosts, networks, network types and network peering points. Apart from this enormous complexity in the infrastructure, these developments have also led to the situation where there is not the cross-ISP communication required for engineering or debugging of network performance problems and security incidents. Above that, competitive providers, operating at relatively low profit margins, did not place a very high priority on gathering or analysing data on their networks, and rather focused on meeting the growing demands of their customers and on additional capacity [CL99]. As a result empirically grounded research in wide-area Internet modelling has not been getting enough attention [CL00].

This, in turn, resulted in the situation where today's Internet industry lacks the ability to evaluate trends, to identify and analyse performance problems beyond the boundary of a single ISP, or to prepare systematically for the growing expectations of its users. Two phrases from articles by K.C. Claffy are worth citing in this context. "Historic or current data about traffic on the Internet infrastructure, maps depicting the structure and topology of this amorphous global entity, or projections about how it is evolving, simply do not exist" [CL99]. "While the core of the Internet continues its rapid evolution, measurement and modelling of it progress at a leisurely pace" [CL00].

Relevance

Being able to identify and understand bottlenecks and a firm understanding of the performance of the global Internet and how this is evolving is very important in this time, where an evolution towards packet switching for all types of information services is taking place and where a lot of new (multimedia) services and products are being deployed and offered, depending on the Internet, or at least on TCP/IP-techniques. The quality of these new services often depends on the performance, so insight in this area will aid in the successful deployment of these new services, such as Internet telephony and multimedia services.

Roughly, the idea of Quality of Service (QoS) in the Internet is that transmission rates, error rates and other characteristics can be measured, improved, and, to some extent, guaranteed in advance. QoS is of particular concern for the continuous transmission of high-bandwidth video and multimedia information [Whatis]. Internet telephony, for example, becomes unacceptable when the end-to-end delay exceeds 100ms although some loss may be suffered. Therefore bandwidth has to be reserved, or at least enough capacity has to be available. On the other hand, in non real-time applications such as file transport, some delay may be suffered, but loss has to be kept to a minimum. This means some mechanisms have to be available to prevent loss. Both these examples, though focussing on different aspects of QoS, show that in order to be able to guarantee some Quality of Service some knowledge of the current state of the network and of the development hereof is necessary.

In the light of QoS where some capacity or bandwidth has to be reserved some enhancements in network functionality are needed. Examples of these enhancements are functionality's such as multiple qualities of service and bandwidth reservation systems such as Resource Reservation Protocol (RSVP). This will at least require some mechanisms for accounting for network usage. [MoCl96] But even if no resource reservation will be made, end-to-end performance measurements will be valuable for making sure enough capacity will be available at any given time and thus maintaining a certain QoS for the end-users. Again, having some means to measure the loss-rates or delay on some link and developing models to predict these figures can help improve the QoS.

On top of this, more and more user communities, like higher education and research communities, the financial sector, industries and others, now view the Internet as the appropriate medium for their future communication need. This will put increasing pressure on the ISP's to collect, analyse and share data related to Internet (and provider) performance.

The automotive industry (AIAG), for example, has a Telecommunications Working Group, in which they are working to set up necessary requirements for the quality of service necessary to satisfy their industry's need. The AIAG has identified several metrics which it views as critical, among which are performance metrics such as latency, packet loss, link utilisation and BGP4 configuration and peering arrangements, as well as reliability metrics such as physical route diversity, routing protocol convergence times and exchange point availability [MoCl96].

These and other developments make it clear that end-to-end performance measurement can benefit both users and ISP's in the light of a better Quality of Service, which in turn makes better and more reliable new services possible.

When we look at another area closely related to QoS, network design and operations, measuring the properties of a wide range of network paths facilitates research in how the global network behaves and evolves [PMAM98]. This in turn can lead to better performance of the own network. Also insight will be obtained in the critical role that a single backbone, traffic exchange point and even individual routers play in the total Internet traffic [CL99]. This may lead to a better understanding of bottlenecks. This in turn can aid ISPs in their design decisions and in identifying and neutralising bottlenecks, for example by installing extra capacity or bandwidth. So, we see that consistent measurement of well-defined metrics can lead to better and more efficient network design and operations in the global Internet as well as in the individual ISP’s networks. Also theories, models and simulations used in network design will provide us with some data or figures about estimated or current performance or traffic. These figures, in turn, can be verified by the measurements from the real Internet. Thus we can use measurements to fill our models, but also to verify our models.

One of the areas in which measurements are broadly accepted is in the area of benchmarking of ISPs. This benchmarking can be used to compare ISPs on certain criteria and base your choice for a certain ISP upon that benchmark. A different aspect of ISP performance comes in to play when we look at Service Level Agreements (SLA). It is clear that if ISPs can guarantee some degree of quality to their users or connecting ISPs, they will have a competitive advantage. However the whole concept of Service Level Agreements (SLA) requires some mechanism to be able to verify the delivered service by the ISP’s., both by the ISP itself as its customers. Again, measurements and measurement infrastructures which truly measure the end-to-end performance can provide these mechanisms and thus play an important role.

Why end-to-end?

As long as services only involve data packets that originate and destine within the boundaries of one network or within the scope of one Internet Service Provider (ISP), the gathering of the required information will be straightforward. At this point however, it is important to realise that in delivering services over the Internet the data almost always has to traverse the networks of multiple (smaller and bigger) ISP's. Again, Internet telephony will only have real value once you can call people across the globe and not just within the network of one operator. This example stresses the importance of having insight in the end-to-end performance of the global Internet in relation to the QoS. Obtaining this insight is an important precondition for the successful development and deployment of new QoS dependant services.

Another area, closely related to QoS, where insight in the end-to-end performance and in bottlenecks is needed is in network design and operations. To be able to optimise a network some statistics on performance are needed. Most ISP's collect some basic statistics on their own network's performance and traffic flows, such as throughput, availability and delay statistics. The problem with these measurements is that the only baseline against which network operators can evaluate their performance is against their own past performance. There is no data available against which global level comparisons or comparisons with other networks' performance can be made. This calls for information on end-to-end performance, information which is beyond the realm of what is controllable by individual networks [MOCL96]. Above that, no scalable mechanisms are available for tracking and resolving problems originating or extending beyond the control of an individual network [MOCL96]. Still these problems could be the main bottleneck in some slow connections or the main cause of low QoS of certain services, and therefore need to be tracked down and solved in order to keep the performance at an acceptable level.

A third area, also closely related to the subjects described earlier, where insight in the performance and in bottlenecks can prove useful is routing. Today's routing algorithms in the Internet are based on shortest path methods. By gaining insight in the geographical locations or other properties of networks paths (such as delay and loss figures) better informed routing decisions could be made. It might even be possible to design routing protocols where the routers are able to adapt to the state of the network as a whole in a better and more efficient manner than is common today. Routers that adapt to certain states of the network will probably be better in a prompt delivery to the end-user, and thus provide higher QoS. Adapting to the state of the network, however, doesn't necessarily imply that different routes will be chosen. Sometimes, for example, it might just be better to wait until the brief congestion on the chosen route is dissolved, instead of changing the routes as soon as some congestion or other problems are detected. This is because routing instability can also have a major affect on the performance of individual networks [MOCL96].

The previous (incomplete) description of problem areas shows the importance of gaining insight in the end-to-end performance of the Internet and in bottlenecks. This leaves the question of how this insight can be gained, which will be discussed in the next section.

Internet Measurement and modelling

Insight in the end-to-end performance of the Internet can be gained by measurement of performance characteristics, such as delay and loss-rate, and modelling of the measured behaviour. This way we can try to explain why certain connections have certain characteristics, what parameters are important and how network behaviour can be predicted. This will thus increase our understanding of the performance and of bottlenecks.

We can distinguish at least two ways to measure the characteristics of connections in a network. **Passive** or workload measurements involve the collection of traffic information from a point within a network, e.g. data collected by a router or switch or by an independent device passively monitoring traffic as it traverses a network link. workload measurement.

Active performance measurements involve the introduction of traffic into the network for the purpose of monitoring delay between specific end-points [CL99]. In active measurements the test traffic can be generated under well-defined and controlled conditions, whereas passive measurement depends on the traffic that happens to pass a certain point. Thus if we want to deliver an objective measurement of the performance of the Internet, active measurements are preferred [UKKW98]. A disadvantage is that active measurement might influence the performance of the network, since it introduces extra (measurement) traffic. Also some device is needed to produce the active measurements.

There are currently many independent studies in progress to measure and analyse the end-to-end performance of the Internet, and this subject has proven to be very difficult. Most of the research in this area is based on traditional tools and only measure from one perspective or from only a few hosts. These traditional tools often use the ICMP- or UDP-echo messages to measure the so called Round Trip Times (RTT) from and to a particular machine. These messages are often ignored or are treated with a very low priority by routers, so the information from these tools has very little significance

[VR00]. Above that, these kinds of measurements involve a large number of parameters that are difficult, or maybe impossible, to model independently [CL99].

On top of this, an Internet performance study by V. Paxson shows, among other 'pathologies' of the Internet, that RTTs have little meaning due to asymmetries in network paths[PA97]. He also demonstrates that the global clock synchronisation by the NTP protocol is insufficient to facilitate proper measurements. Inspired by Paxsons results, the 'IP Performance Measurement' working group [IPPM] has developed well-defined neutral performance metrics with broad acceptance, such as '**one-way packet delay**' and '**one-way packet loss**'. A framework for IP performance metrics by the IPPM can be found in [PAMM98]. These metrics are based on active measurements and thus some device or infrastructure is needed to generate these measurements.

In an effort to support these more standardised measurements, some research groups are trying to deploy some measurement technology and infrastructure. They mainly focus on the performance and reliability of selected Internet paths, and try to determine what specific segments of a given path limit that performance and reliability [CL99]. Two of these efforts are the RIPE Test Traffic [RIPETT] project by RIPE-NCC, mentioned earlier, and the Surveyor [KZ99] project by Advanced Networks & Services. These projects use dedicated and accurate measurement boxes which actively send measurement packets to each other. The test boxes of the RIPETT project are placed at about 40 Internet Service Providers (ISP), mainly in Europe, and those of the Surveyor project are placed at universities in the American part of the Internet. The objective of both measurement undertakings is to provide a service to the collaborating ISP's to monitor anomalies and errors in their network connections [VR00].

The data from these projects can also be very useful in modelling Internet performance, which is one of the major goals of the UMEEPI project. When building models of the Internet performance the goal is to predict certain parameters based on other, known parameters. One such effort would be to predict the delay on a certain path based on the number of hops in this path, an example of which is given in [MHH00]. This study will be briefly addressed to in the next paragraph. Another example of modelling internet performance is to try to obtain delay information for individual links in a graph based on the end-to-end delays between multiple end-points. This study has been performed by another student within the UMEEPI project [SAMSON].

Demarcation of the problem

In this paragraph the scope of the research effort is described. We move from the broad subject of internet measurement and modelling and the problems described earlier to the more specific subject of topology maps at the AS level in several steps. These are in fact the steps that where taken to arrive at the formulation of the problem as described in the next paragraph.

Topology

By now, we are convinced that measuring, analysing and modelling of certain performance metrics can be very useful. One problem with these kinds of study is that little is know about the characteristics of the network under investigation, the Internet.

A lot has been said and written about the performance of networks. Many theories, models and simulations are based on certain assumptions, for example assumptions about the interconnection graph of the network. Some theories concerning optimal routing are based on the assumption that we completely know the graph and other theories build on a random interconnection graph. The problem with the Internet topology, however, is that it is not fully known, but, as far as we know, it also is not random. Insight in the real physical interconnection can be used to validate existing or new theories, models and simulation of Internet traffic. This stresses the importance of gaining insight in the real structure of the Internet.

When we look at delay, for example, we see that this can occur in every step of a transmission of data, and that we can distinguish different kinds of delay, such as transmission-, propagation-, queuing- and processing delay [STA]. Two of these delays occur in the router that forwards a packet: queuing- and processing delay. It is obvious that the contribution of these delays to the total delay will

be less if the shortest route (less hops) is chosen. Having some knowledge of the probability density function (pdf) of the shortest path hopcount, thus enables estimates of the end-to-end delay in the Internet. This can be used to determine the feasibility of offering service differentiation in the current Internet topology [MHH00]. A first step in determining this pdf lies in understanding the Internet topological structure. However, it needs to be said that the number of hops isn't the only predictor of delay, especially when a five hop route which travels through a satellite (propagation delay) has to be compared to a seven hop route which travels over high-speed links. Recent studies have indeed found no correlation between the hopcount and the delay [MI01]. Insight in the topological structure will aid in addressing this problem

One other area where insight in the topological structure will be useful is in network design and optimisation. Analysing multiple paths from different vantage points within one network can help diagnosing the performance problems in the interior of the network. If we can visualise path data, this would aid in the troubleshooting of the own network of an ISP, but also in the troubleshooting of the entire measured part of the Internet.

Apart from these practical uses of insight in topology it can also help in the modelling efforts of the UMEEPI project. The robustness and reliability of the Internet are dependent of efficient and stable routing between the networks of different providers. Once we have a topology map we can try to extend this to the dynamics of paths and physical connections and to modelling hereof. Then we can relate effects of topology changes to the end-to-end performance. For example when we see a path being redirected at a certain time we can see what kind of effects this has on the end-to-end performance. We may then also be able to simulate parts of the network and try to foresee what kind the effect will be of changing routing policies on the performance and connectivity. The predictions can later be verified by the measurements which in turn will aid in adjusting the models. This also greatly enhances the possibility for individual networks to react to policy and topology changes of other ISP's

Note that although we're talking about a graph, this doesn't mean a drawing has to be made. There are different ways to describe a graph and one of them is using a matrix. This graph can be used to compare current delay and loss figures to past figures. Once detectable changes in delays or loss are found they might be explained by detected changes in the path between two end points.

We see that reconstructing the interconnection graph from the available traceroute data can provide us with some valuable insights. If we can visualise the topology of the Internet this will eventually aid us in the identification of bottlenecks and in analysing packet dynamics. Also visualising the graph of the Internet at a certain point in time is a prerequisite for visualising the changing map of the Internet and thus for showing the dynamics. Insights in the interconnection graph also gives us an idea of the applicability of theories based on the assumption of a random interconnection graph. Also we could apply graph theory to gain more insights.

So, it is clear that insight in topology might help us in our modelling effort, and in fact the project proposal says the following about this. To be able to model characteristics of the end-to-end performance along a certain path we need to gain some insight in the topology of the network [Vr00]. We see that mapping IP-addresses to more useful entities, such as autonomous systems, router equipment (multiple IP addresses can refer to the same machine) and geographic location and determining the physical connections and representing these in some way are the two main challenges in this area.

Different aggregation levels of topology maps

When we talk about topology data, we talk about a description of the network link infrastructure at a variety of 'protocol' layers. This means we can try to determine the topology at the IP level (which paths are available?) or, for example, at the physical level (which routers are physically interconnected?). A study of this topology at some level of abstraction is necessary in order to gain a firm understanding of how the network performance is influenced by topology changes and how measured behaviour can be explained. Gaining insight in topology thus is an important step in the modelling efforts of the UMEEPI project.

With a graph of real physical interconnection or chosen paths available we might be able to determine optimal paths using some routing algorithms (such as shortest path) and compare those paths to the real chosen paths (from the same traceroute data) and thus determine whether routing in the Internet is optimal or not. This method can be extended when some metrics (from the delay- and loss- figures) are placed along the links. This makes it possible to determine optimal weighted paths and compare these to the chosen paths.

Another example is a graph that shows the different routers and the physical connection between them, possibly even on a geographical map if some additional information concerning location can be obtained or estimated.

However, it remains the question whether this physical interconnection graph can be obtained from the experimental data. To answer this question we have to think about the information which is produced by the traceroute tool. Instead of insight into the real physical interconnection these paths provide insight into the routing within the network.

This brings us to the question whether routing information isn't far more interesting than the real physical structure. The traceroute data provides us with the chosen paths at a certain time, and though this may not directly give insight in the real physical interconnection, it does show how traffic is routed through the network at a certain time. In the end this is what matters when looking at the end-to-end performance.

However, it might still be possible to gain some insight into the real physical connection. We know that between two routers that are subsequent in a traceroute path a physical connection has to be available. In order to gain insight in the physical interconnection a technique similar to medical x-ray tomography can be used. Here a three dimensional image is achieved by rotating an x-ray emitter around the subject. Geologists rely on similar techniques to build models of seismic activity using cross-section images (slices) of the earth. Data gathered from tomographic scans play an important role in developing models to analyse and predict certain phenomena [Cl99].

By looking at the network form different vantage points, we do something similar and we can possibly reconstruct the real physical structure up to a certain point. The one thing we don't know is whether there is a physical connection between two routers that aren't subsequent in any traceroute. In other words we can never rule out additional connections, because the fact that we don't see them, doesn't mean they're not there. The fact that an available physical connection is never chosen and thus doesn't show up in the traceroute data can probably be explained by the fact that the connection is slow, or not very reliable and thus is not chosen by the routing algorithms. But it might also be possible that a wrongful configuration in one of the routers is the cause of a good connection not being used. These kind of errors are hard, if not impossible to detect merely from traceroute data.

Focussing on autonomous systems level topology maps

Despite the problems described above, graphs of Internet paths at multiple aggregation levels are still important to support our effort to understand the measured behaviour. For example a graph where the connectivity of different autonomous systems is visualised might provide valuable information for tracking possible inefficient routing and bottlenecks. It also might give some information concerning routing policies and provide insight in the critical role of certain backbones or traffic exchange points. However, if we would succeed in determining connectivity of individual machines we would get a graph with a lot of points, which will be difficult to interpret. This makes it necessary to make some aggregation so we can gain a certain amount of usable information without drowning in too many details.

Every host or router in the Internet belongs to an autonomous system, which can be referred to as a portion of a network, usually within the control of one organisation and usually running a single routing protocol. Routing between autonomous systems is done with a protocol that is defined as being an inter-domain or inter-AS protocol or an exterior gateway protocol [RP00]. Each AS has a unique number within the Internet.

The traceroute data from the RIPETT project consists of the subsequent IP addresses in a certain path. If these IP addresses can be clustered or mapped to AS-numbers a major simplification of the data can be made. This way we might be able to draw a graph in which the nodes represent the different autonomous systems and the edges represent the possible paths between them. This will provide insight in inter AS connectivity and possibly in routing policies of different ISPs. On top of this insight in the crucial role of certain backbones or internet exchanges might be gained.

It might be possible to obtain some more information on topology from so called BGP routing tables, which reflect the transit relationships between individual Autonomous Systems at a given point in time [Cl99]. This information might, for example, be used to verify our own mapping, but it might also give some insight in whether advertised paths (in the routing tables) are really used and information like this. Also the robustness and reliability of the Internet are highly dependent on efficient, stable routing among provider networks [CL99], so analysis of the current routing behaviour is of importance. Observations of macroscopic routing dynamics provide insights into [Cl99]:

- Effects of outages on surrounding ISPs
- Effects of topology changes on Internet performance
- Unintended consequences of new routing policies
- Potential areas for improving an individual networks' ability to respond to congestion and topology changes.
- Infrastructural vulnerabilities created by dependencies on particularly critical paths

Therefore looking at inter-domain routing is also very interesting in this context.

Formulation of the problem

In the problem analysis we see that a part of the UMEEPI research lies in gaining some insight in the topology of the measured part of Internet. I will focus on this part because it also can be an important starting point for the other parts of the UMEEPI project. In this chapter the goals and research questions will be stated.

Goals

We've seen that lack of insight in the end-to-end performance of the Internet and the packet dynamics causes several problems. On top of this we've seen that insight in the topology of the network can aid in understanding bottlenecks and end-to-end-performance. The major goal of this thesis study will be to provide this insight into the topology of the network. This, however is still a bit vague and needs to be specified.

Insight in the topology of the network can refer to multiple aggregation levels, ranging from placing individual routers and connections on a geographical map to an overview of the various autonomous systems and the interconnection between them. By looking at the topology problem at the level of autonomous systems we can cluster a lot of different IP addresses into one autonomous system number, and thus simplify the problem of gaining insight in the topology of the measured part of the Internet. This seems like a reasonable starting point.

For insight in the topology of the network to be useful we have to think about ways to represent the data. In general it is common to visualise a network as a graph with nodes and links between them. It depends on the aggregation level what the nodes represent. In our approach of looking at the level of autonomous systems a node can represent one autonomous system and the links represent an interconnection between them. With such a graph we can determine optimal paths using some routing algorithms (such as shortest path) and compare those paths to the real chosen paths (from the same traceroute data) and thus determine whether routing in the Internet is optimal or not. This method can be extended when some characteristics or metrics (from the delay- and loss- figures) are placed along the links. This makes it possible to determine optimal weighted paths and compare these to the chosen paths.

This research effort is threefold:

1. We want to increase our understanding of the topology of the Internet at the level of Autonomous Systems, because we expect this to provide us with valuable insights in the overall end-to-end performance and modelling hereof.
2. In order to do this we have to have a firm understanding of inter-domain routing. We have to gain insight in how this works in theory and how it is applied in practice.
3. Also a tool that maps IP addresses to AS numbers and visualises the interconnection of Autonomous Systems is expected to increase our understanding of the end-to-end performance and to help us in our modelling effort. Not only the visualisation but also characterisation of AS-AS connections, or in other words the usability of our visualisation has to be studied. Once we have the visualisation, what can we do with it?

This results in the following goal for this research effort:

The goal of this research effort is to increase our understanding of the topology of the Internet at the level of Autonomous Systems, to understand inter-domain routing in the Internet and to determine whether it is possible to obtain topology maps at the level of Autonomous Systems, by designing and implementing a prototype visualisation tool.

Research questions

The goal described above results in the following research questions, with several related subquestions:

1. How does inter-domain routing, or inter Autonomous Systems routing in the Internet work, both in practice as in theory?
   1.1. How does the Internet work?
   1.2. What are Autonomous Systems?
   1.3. How does routing in the Internet work?
   1.4. How does inter-domain routing in the Internet work?
   1.5. What is peering?
   1.6. What is policy routing?
   1.7. How do peering and routing policy influence inter-domain routing?

2. Is it possible to design and implement a tool that can map IP addresses to AS numbers and translates the given traceroute data into a topology map of the interconnected Autonomous Systems?
   2.1. What information do we need to map IP addresses onto AS numbers?
   2.2. How can this mapping process be implemented?
   2.3. What problems do we encounter when trying to map IP addresses onto AS numbers?
   2.4. How do we transform traceroute data into AS paths?
   2.5. What problems do we encounter in the RIPETT data?
   2.6. Does geographic placement of AS nodes give additional insight?
   2.7. Can a prototype visualisation tool be designed and implemented?

3. What is the use and relevance of topology maps at the AS level, what kind of information can be derived from these and what are the limitations and problems?
   3.1. How can AS level topology maps be interpreted, what do they represent?
   3.2. How can AS level topology maps be helpful in understanding and modelling the Internet?
   3.3. How can AS level topology maps be helpful for individual ISPs?
   3.4. Can insight in dynamics of inter-domain routing tables be gained from the visualisations?
   3.5. Can we gain insight in routing policies and peering from the AS level visualisations?
   3.6. Can we gain insight in the “Internet Backbone” from the AS level visualisations?
   3.7. What are the limitations of the developed tool and of the AS level topology maps?
   3.8. Is further research into AS level topology maps useful, or are other abstraction levels such as physical level topology maps more promising?

The remaining chapters provide an answer to these and other questions that emerged while exploring the subject. In the final paragraph of this chapter a short research design is given.

Research design

Now that the goal and research questions are clear, the research design can be completed. Here the object of study, the methods, the scope and the planning are briefly described.

Object

A thorough study of Autonomous Systems, IP, addressing, routing protocols such as BGP has to be made. This is not only theoretically but also insight in the current practice of inter-domain routing has to be gained. Another object of study is the implementation of a tool that can represent a graph of the autonomous systems from the available data. This study is highly explorative of nature and therefore probably more new questions are raised than are answered. These questions are left as recommendations for further study.

Method

A lot of the theory can be obtained from the Internet and from books, so large parts of the research effort will involve literature study, but insight in current routing practice involves routing policies and insight herein will be difficult to obtain. However, the implementation of the tool will have priority because it is expected this will allow us to gain major insight in the whole subject. Therefore a large part of this research will be experimenting with traceroute data and programming the prototype tool.

First the traceroute data will be analysed using existing tools after which an initial step towards developing a visualisation tool. At first a tool for IP level visualisations will be developed and the resulting graphs will be studied. After this some attempts are made to implement a mapping process and to visualise the traceroute data at the AS level. Also here the resulting graphs will be studied.

Also knowledge and experience with JAVA programming needs to be developed, so this will require some additional time.

Scope

The research effort described here focuses on topology maps of the Internet at the Autonomous Systems (AS) level. This study tries to determine whether such maps can be derived and tries to give insight in the usability of such maps in understanding and modelling the end-to-end performance of the Internet. To this end a prototype of a visualisation tool has been developed and some topology maps have been made from the data. These topology maps are studied to gain insight in the usability of AS level topology maps and to determine whether further research into AS level visualisation and modelling is promising.

Time

This study has been performed part time for 3 to 4 days a week during the period from October 1st 2000 to October 1st 2001, including the writing of this final report.

2. The Internet

In this chapter some of the concepts and terminology of the Internet are discussed to provide us with a common understanding and vocabulary. First some common terminology in discussing computer networks and internetworking is briefly recapitulated, followed by a brief introduction to the concept of Layering. After these two introductory paragraphs the focus will shift to the Internet.

A brief history of the Internet is given, because some of the concepts that were designed in the early years of the Internet are still used today, which sometimes is the main reason for some of the problems that we currently face. An example of this is the 32-bit address-space that provides us with about 4 billion IP addresses, which seemed plenty initially. A slightly larger address space would have greatly increased the number of addresses thus eliminating the current address shortage problem. Also the two-level hierarchy with the concept of autonomous systems used in routing today has an historic background.

After this description a more technical description of the layers in the Internet protocols will be given. The focus here will be on the network layer protocols since this layer is responsible for routing of packets from source to destination, which is the main subject throughout this study. The network layer protocol IP (Internet Protocol) is the glue that holds the Internet together [TAN96].

Computer Networks

In this paragraph a brief recapitulation of the basics of computer networks is given to provide us with a common understanding of the used terms and concepts. For a more in depth coverage of these subjects see [TAN96].

The term "computer network" can be defined as an interconnected collection of autonomous computers. Two computers are interconnected if they are able to exchange information. The term autonomous rules out systems with a clear master/slave relation or computers with remote printers and terminals from our definition. A **host** is a computer whose primary use is directly for an end user or as some kind of server. All the traffic arriving at a host is intended for that computer and not for some other device [ST98]. With a **router** however, things are different. Here traffic that arrives at the device is likely to be intended for some other device and this traffic needs to be routed to this other device. More about routers later in this chapter and in Chapter 3. Among the goals and uses of computer networks are the ability to share resources, access to remote information, person-to-person communication and interactive entertainment [TAN96].

There are multiple dimensions along which networks can be classified [TAN96]. One of these dimensions is transmission technology, in which we can distinguish broadcast and point-to-point networks. With **Broadcast networks** there is one communication channel that is shared by all machines on the network. Every packet is placed on this communication channel and therefore every machine receives the message. A packet contains an address field specifying the machine that actually needs to process the message. With multicasting only a subset of all systems receives the message. **Point-to-point** networks consist of many connections between individual pairs of machines. On the path from source to destination, a packet may need to travel multiple machines. Often different paths of different length or cost are possible, so routing algorithms play an important role in point-to-point networks. Within the Internet both principles are possible but at the Internet Layer the concept of point-to-point holds. More information about the Internet Layer will be given later on.

Another dimension is the scale of the network. There are roughly three different types of networks in this context. Especially the description of a WAN is important, because some important concepts of the Internet and IP are mentioned there. The other two types are mentioned for completeness:

1. **Local Area Networks (LANs)** are privately owned networks that cover a relatively small geographic area. Due to the restrictions in size the worst-case transmission time is bounded and known in advance. Two common topologies are the bus (Ethernet) and the ring (IBM Token Ring) topology.

2. **Metropolitan Area Networks (MANs)** are basically bigger versions of LANs and use similar technology.
3. **Wide Area Networks** (**WANs)** are networks that cover large geographic areas and often use transmission facilities provided by common carriers, such as telephone companies. WANs contain a collection of hosts that are connected by a **communication subnet**, or just subnet[1]. The communication subnet carries messages from host to host and consists of two basic components: transmission lines (**circuits, channels, trunks**) and switching elements (**routers, packet switching nodes**). There are no hosts inside the communications subnet, only the infrastructure that "gets" traffic from its source to its destination is meant with the term, see Figure 2-1.
   If two hosts or routers that do not share a cable wish to communicate, they must do so indirectly, through other routers. This often goes using the **point-to-point**, **store-and-forward**, or **packet-switched** principle. Every packet finds its own way to the destination and travels from router to router, where packets are received, stored and forwarded on the desired output line. This can cause packets to arrive in a different order than that in which they were sent. Figure 2-1 shows the relation between the host, the router and the communications subnet.

The **topology** of a network is the way the routers are organised. When a point-to-point communications subnet is used this topology is an important design issue. This topology can often be represented as a graph. Some possible topologies for a point-to-point communications subnet are a star, a ring, a tree or a complete topology, were every host connects to all other hosts. Most LANs have some symmetric topology and most WANs are irregular.

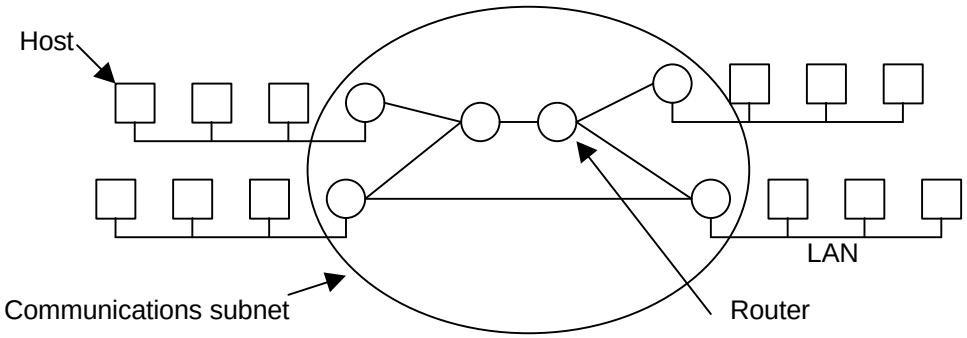


Figure 2-1. Host, Communications subnet and Router

A collection of interconnected networks is called an **internetwork** or just **internet**. Different and often incompatible network can be connected, often using **gateways** to make the necessary translations. A common form of an internet is a collection of LANs connected by a WAN. In this case the network looks like the one in the Figure above, with the term communications subnet replaced by WAN. However, if no hosts are active in the network that connects the LANs we speak of a subnet and not of a WAN, but we still have an internet, because multiple LANs are connected. The world-wide Internet is an example of an internet.

The Concept of Layering

The concept of layering is widely used in the networking world. Also the literature on this subject is extensive. However, because the concept of layering is a useful framework for explanation and discussion, the basics of layering will be briefly repeated here.

To reduce the complexity of network software, most networks are organised as a series of **layers** or **levels**, each one built upon the one below it. The number of layers and the name, contents and

[1] Note that the term subnet is also used in relation to network addressing and that it has a different meaning in that context. Therefore, we will use the term “communication subnet” in this context. See chapter 3 for an explanation of the other meaning of the word subnet.

function of each layer differ from network to network. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented. The rules and conventions used in a conversation between the same layer *n* (**peers**) on different machines are known as the layer *n* **protocol**. A protocol is a set of rules governing the format and meaning of the frames, packets, or messages that are exchanged by the peer entities within a layer. In other words, it is the peers that communicate using the protocol. Between each pair of adjacent layers there is an **interface**. This interface defines which primitive operations and services the lower layer offers to the upper one.
A set of layers and protocols is called a **network architecture** and a list of protocols used by a certain system is called a **protocol stack** [TAN96]. Figure 2-2 shows the relation between layers, protocols and interfaces.

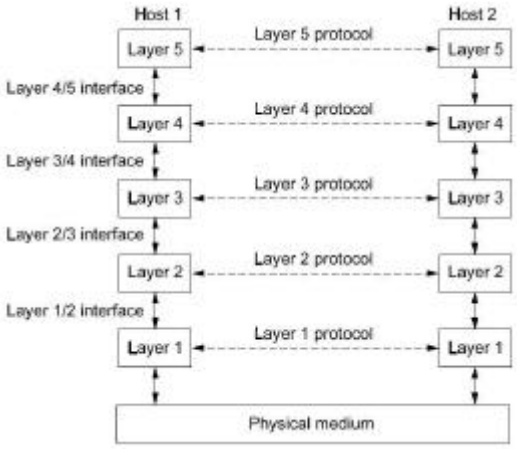


Figure 2-2. Layers, protocols and interfaces [TAN96]

As a message goes down in the protocol stack in the left machine, each layer *n* ads its own header, containing information intended for the layer ***n*** software in the right machine. Among the information in such a header will be the address of the right machine at that specific layer. As we will see in Chapter 3 when a router receives a packet, this is because the packet had the physical layer address of the router. Upon reception of this packet, the router peels of the physical layer header and inspects the network layer header, for an IP address. When it finds this IP address and when it is not intended for the router itself it will add a new physical layer header with the address of the "next hop" and copy the new packet to the desired interface. This will be explained more in depth in Chapter 3. The point here is that the right machine peels of these headers as the message travels up through the protocol stack.

In general, the service a networking protocol provides to the layer above it can be characterised as being either connection-oriented or connectionless. **Connection-oriented** data handling involves using a specific path that is established for the duration of the connection. **Connectionless** data handling involves paths that can be different for different packets in the same communication. Because the packets take different routes, they can arrive out of order.
Another characterisation of the services delivered by the layer to the layers above them is the reliability, often in the sense of that they never lose data. This reliability is usually implemented by using acknowledgements after the reception of each message. An example of an unreliable connection-oriented service is the unreliable connection, where indeed a connection is set-up, but no acknowledgements are used because of the overhead and delay this produces. This is especially useful for services such as telephony or video, because it is better to miss lose some data and to preserve the flow of data, than to stop and wait as errors are being corrected. An example of a reliable connectionless service is the acknowledged datagram service.

From ARPANET to the current Internet

In this paragraph a brief description of the history of the Internet is given, because this gives insight in how the Internet was designed and how it works. This also gives insight in the loose way the Internet is organised, which makes it difficult to control and secure the network, but which also leads to immense possibilities [DHHLS98]. In the next paragraph we will look more closely at the different layers and the most important protocols at those layers in the current Internet.

In response to the launch of the Sputnik in 1957 by the Soviets, the US forms the Advanced Research Projects Agency (ARPA), within the Department of Defence. The goal of ARPA was to establish a lead in science and technology applicable to the military [ZA01]. The first head of ARPA (DARPA at that time), J.C.R. Licklider of MIT posed the “Galactic Network” concept, in which he envisioned a globally interconnected set of computers through which everyone could quickly access data and programs from any site [ISOC00]. This idea was much like the current Internet. Licklider convinced his successors at DARPA, Ivan Sutherland, Bob Taylor and MIT researcher L.G. Roberts, of the importance of this networking concept.

Leonard Kleinrock at MIT published the first paper on packet switching and in the July 1961 [KL61] and the first book in 1964 [KL64]. Kleinrock convinced Roberts that packet switching had to be the choice for the envisioned network, and in 1966 Roberts went to DARPA to develop the computer network concept and he put together his plan for the “ARPANET”. Simultaneously the RAND group had written a paper on packet switching networks for secure voice in the military [BAR64]. Although it is commonly believed that this networking concept was designed to survive a nuclear strike of some sort, this RAND paper, unrelated to ARPANET, was the only document that actually talked about nuclear war. Still the later work on Internetworking did place emphasis on robustness and survivability in case of (temporary) loss of hardware and links.

In 1969, the first packet switch, the Interface Message Processor (IMP) was installed at UCLA and the first host computer was connected. Soon, when Stanford Research Institute was connected to the ARPANET, the first host-to-host message was sent from UCLA to Standford. Later that year two more nodes were added at UC Santa Barbara and University of Utah and at the end of 1969, four host computers were connected together into the initial ARPANET. Figure 2-3 shows two well-known sketches of this early network.

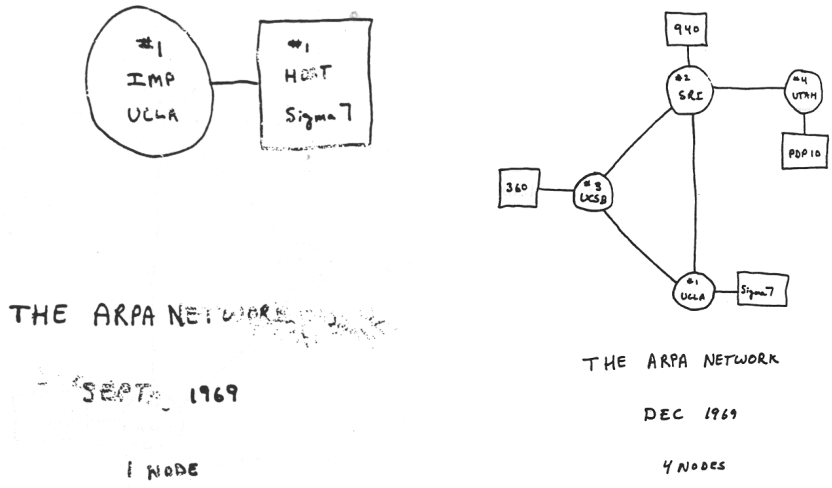


Figure 2-3. The first IMP-host connection and the 4 nodes at the end of 1969 [ZA01].

Computers were quickly added to the ARPANET during the following years and with the completion of the host-to-host protocol (NCP) the users of the network could begin to develop applications on top of this protocol. In 1972 email was introduced, which instantly was the largest network application for the next decade [ISOC00].

When more and more university and government networks got connected and finally satellite and radio connections where established (from the packet radio project at DARPA), it became clear that the ARPANET protocols where not suitable for running over multiple networks. This led to more research in to protocols, which in turn led to the invention of the TCP/IP model and protocols in 1974 by Cerf and Kahn. TCP/IP was specifically designed to handle communication over internetworks, which obviously became more and more important [TAN96].

Some of the basic thoughts on which this work of Cerf and Kahn was based were that each distinct network would have to stand on its own and that no internal changes would be needed to connect it to the internet. Communications would be on a best effort basis, black boxes (later gateways or routers) would be used to connect these networks and there would be no global control at the operations level [ISOC00]. These and other concepts have led to the current strength, but also to the current problems of the Internet. For example, the internet was never designed with secure banking transactions in mind, but today everybody seems to want these kind of services.

At that time also the 32-bit address space was designed. Since the enormous growth of the number of LANs and of the number PCs connected to the Internet could not be foreseen at that time this seemed like a reasonable design choice. Because the first 8 bits where used to identify the network, a total of 256 networks could be connected at that time.

In 1983, the ARPANET consisted of over 200 IMPs and hundreds of hosts and it was still stable and successful. In that year ARPA turned the management of the network over to the Defence Communications Agency (DCA), that split it into a military part (160 IMPs), MILNET, and the rest. Stringent gateways where placed between MILNET and the remaining research communications subnet [TAN96].

This ongoing growth led to the situation where finding a host costs more and more time and money. In response the **Domain Name System** (DNS) was created to organise machines into domains and map host names onto IP addresses. Since then DNS has become a distributed database system for storing a variety of information related to naming [TAN96].

By 1990, the ARPANET was dismantled, because larger and faster networks had emerged from it. NSFNET is one of these networks that was designed in 1984 by the NSF as a successor for the ARPANET. The NSF build a high-speed (56 kbps at that time) backbone between 6 supercomputers. The switching nodes, called “fuzzballs”, spoke TCP/IP from the start, making NSFNET the first TCP/IP WAN. NSFNET was connected to ARPANET by a fuzzball – IMP connection. Due to the enormous success of NSFNET the backbone was upgraded to 1,5 Mbps, later upgraded to 45 Mbps and the goal was to upgrade to 3Gbps by the end of the millennium.

By 1995 the NSFNET backbone was no longer needed to interconnect the NSF regional networks, because many companies were running commercial IP networks. When it was sold to America Online in 1995 (running at speeds up to 3 Gbps) the regional NSF networks had to buy connectivity from commercial IP networks. To ease the transition and to be sure that every regional network could connect four different Network Access Points (NAPs) where established. Every network that wanted to provide backbone service to the NSF regional networks had to maintain a connection with all four NAPs. Thus a packet travelling from some regional network to another regional network had the choice between competing commercial internet backbones. In addition to the NSF NAPs other government NAPs and commercial NAPs where created and the concept of a single Internet backbone was replaced by a commercially-driven competitive infrastructure[TAN96].

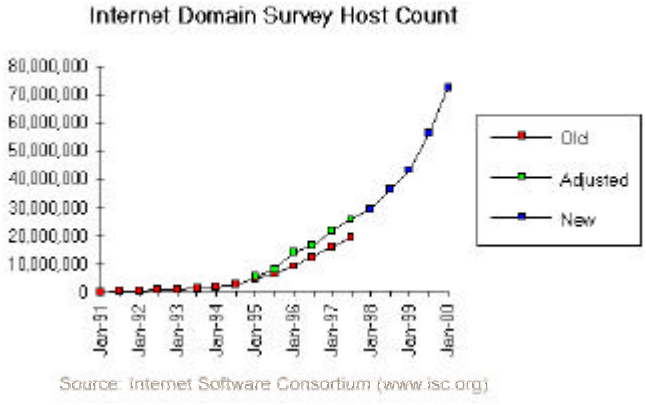


Figure 2-4. The growth of connected hosts

Meanwhile in other countries and regions the same occurred and today there are many, many different “internet backbones”. So many that I don’t prefer the term internet backbone, since this suggest the existence of a single large network that handles all traffic travelling between regional

networks. Instead there are many, competing networks which makes it even more clear how complex the current Internet is organised.

From there on Internet developed into the largest computer network in the world. The growth of Internet, into which ARPANET and later NSFNET developed, is estimated at 80% every year. According to a survey by the Internet Software Consortium (www.isc.org), held in January 2000, there are 72,398,092 hosts on the Internet. Figure 2-4 shows the growth of the number of hosts since 1991.

The Federal Networking Council (FNC) has accepted the following definition of the Internet in 1995:

"Internet" refers to the global information system that -- (i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein.

The Layers in the Internet

One of the most used reference models for describing and designing networking software is the **OSI (Open Systems Interconnection) reference model**. This model can be used to describe a variety of different architectures, but the network architecture of the Internet does not map well into the seven layers described in the OSI-model. Instead we use the TCP/IP reference model, named after its two primary protocols. TCP/IP is the network architecture used in the Internet. For a very good description of both the OSI and the TCP/IP reference model and for a comparison between the two models, see [TAN96].

The TCP/IP model finds its origin in the ARPANET, the predecessor of the well known Internet, were the ability to connect multiple networks together in a seamless way was one of the major design goals from the very beginning. Another design goal of the Department of Defence, who sponsored the ARPANET, was that the network would be able to survive loss of communications subnet hardware. This way existing conversations wouldn't be broken of and communications would be possible as long as the source and destination machines were still intact and at least one path was available between them. Furthermore, a flexible architecture was needed, since applications with divergent requirements were envisioned [TAN96]. Figure 2-5 shows the layers in the TCP/IP network model.

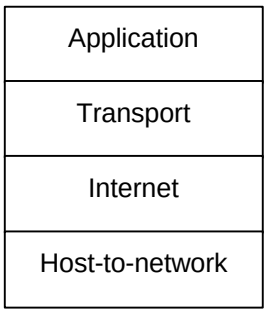


Figure 2-5. The TCP/IP reference model

Host –to-network layer

The lowest layer, **Host-to-network layer** provides the connection of the host to the physical network. This protocol is not defined and differs for different hosts and networks. We will not explore this layer here any further.

Internet layer

The requirements stated in the previous paragraph led to the choice of a packet-switching network based on a connectionless internetwork layer. This layer, called the **Internet layer**, is the linchpin that holds the whole architecture together [TAN96]. The Internet layer allows hosts to inject packets in any type of network and lets them travel independently to the destination. This means that packets could arrive in a different order than that in which they were sent, in which case it is the job of the higher layers to rearrange them, if in-order delivery is desired.

The Internet layer defines an official packet format and protocol called **IP** (**Internet Protocol**) [RFC791]. The job of the Internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly the major issue here, which means that also the addressing system, using IP addresses, is defined at this layer. Also avoiding congestion is part of this layer[TAN96], however this is of less importance to this thesis study.

Every interface of a host or router that is connected to the Internet has a unique 32-bit IP address. Machines connected to multiple networks, such as routers, must be assigned a unique IP address for each network interface. Originally, each IP address has two parts, the first part, the **network prefix**, indicates the network on which the host resides, and the second part identifies the particular host on the given network. The Internet numbering authorities assign the network numbers, so they are unique world-wide. Two hosts on the same network have the same network prefix, but must have a unique host-number. Also two hosts on different networks have a different network-prefix, but may have the same host-number [CS96]. Therefore all IP addresses in use are unique.

To make IP addresses more readable for humans they are often expressed in “dotted-decimal notation”, in which each 32-bit address is divided into four 8-bit (byte) parts and each part is independently calculated as a decimal number [CS96,TAN96].

An IP packets header consists of 14 fields, of which 3 fields are of particular interest to our research:

- **TTL**: this field is used to control the time that a packet will be in the network (time to live). Originally it was designed to count the number of seconds that a packet would be active, however, in practise it counts the number of hops, since each router decrements the value in this field by one. When this field reaches zero, the packet is discarded and a warning packet is sent back to the source IP address. This field is used to prevent packets from moving through the network infinitely and causing enormous congestion, for example when there is a routing loop [TAN96]. This field is interesting to our research, since the tool traceroute, which will be explained later, uses this field in a smart way to get insight in the route taken by a particular packet at a specific moment in time.
- **Source IP address**: This 32-bit field specifies the IP address of the end node that sent the packet.
- **Destination IP address**: This, also 32-bit, field specifies where the packet should be delivered.

Along with the IP protocol, the Internet has several control protocols used in the network layer, like ICMP. ARP (used for mapping IP addresses onto data link layer addresses), RARP (used for discovering the IP address when the host knows its data link layer address) and BOOTP (the same as RARP, but using UDP, so it can be forwarded over routers) are also used, but of less importance to our research.

ICMP stands for **Internet Control Message Protocol** and it defines a series of message types, which are used to report some unexpected events to a host [RFC792]. Some of the most important message types are[TAN96]:

- The **destination unreachable** message is used when the communication subnet or a router cannot locate the destination. Most often some router sends this, but sometimes it is possible that the destination hosts returns this message. This is the case if some higher level service-access point[2] is unreachable. Also when the DF (don’t fragment) bit is set inside the IP packet header, and when a “small-packet” network has to be traversed to reach the destination, this message is returned.

[2] A service access point (SAP) can be seen as an interface between two services in adjacent layers in a given machine.

- The **Time Exceeded** message is sent when a packet is dropped because its TTL has reached zero.
- The **Echo Request** and **Echo Reply** messages are used to see if a given destination is reachable and alive. Upon reception of an Echo Request, the destination is expected to send an Echo Reply message back.
- The **Timestamp Request** and **Timestamp Reply** are similar, but now the arrival time of the message and the departure time of the reply are recorded in the reply. This is used for measurement of network performance.

Although ICMP is presented and defined as a separate protocol, every machine implementing IP is required to implement ICMP as well.

Transport layer

The next layer above the Internet layer is called the **Transport layer**. This layer was designed to allow peer entities on the source and destination hosts to carry on a conversation. Two end-to-end protocols are defined in this layer [TAN96]:

- **TCP** (**Transmission Control Protocol**) is a reliable connection oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the Internet. TCP handles things like fragmentation, sequencing and flow control.
- **UDP** (**User Datagram Protocol**) is a unreliable connectionless protocol for applications that provide their own flow control and sequencing capabilities. UDP is also used for one-time request-reply queries and for application in which prompt delivery is more important than accurate delivery, such as transmitting speech or video.

Application layer

The highest layer is the **Application layer** which contains al the higher level protocols such as telnet (virtual terminal), FTP (file transfer) and SMTP (e-mail). Also DNS is an application layer protocol, which maps variable-length ASCII names (like www.ripe.net) into 32-bit binary IP addresses.

Figure 2-6 shows some of the protocols used in relation to their layer in the TCP/IP reference model.

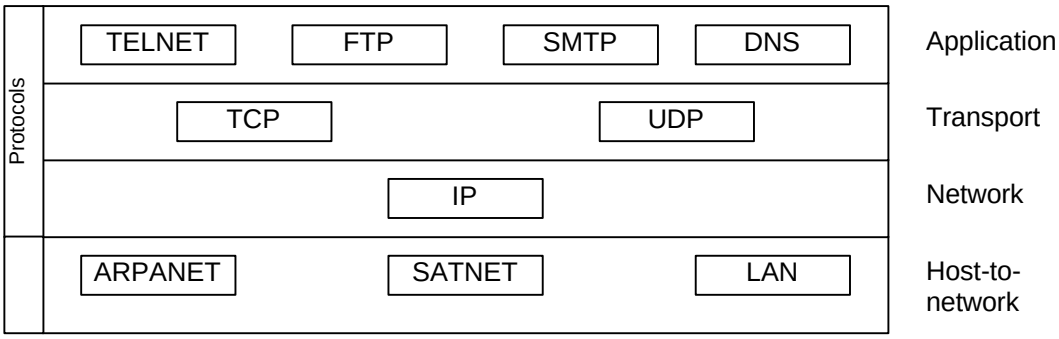


Figure 2-6. Protocols and networks in the TCP/IP model

3. Routing in the Internet

As we have seen, the network layer is the lowest layer that deals with end-to-end transmission and the main function of the network layer is routing packets from the source machine to the destination machine. Most often packets will require multiple hops before the destination is reached [TAN96]. The routers in the communications subnet, using routing algorithms and routing protocols accomplish this task. Before we go into detail about autonomous systems and BGP, we look at routing, routing algorithms and routing protocols in general and at routing in the Internet in particular. Next we will focus on the concept of interdomain routing and autonomous systems. Then a reasonably in-depth coverage of BGP4 will follow, since learning how interdomain routing in the Internet works is one of the goals of this thesis study. At the end of this chapter some considerations and examples of routing policy and peering are given.

Routing basics

In order for a network to work properly, in a sense that every node (host or router) is able to communicate with every other node, the intermediate nodes between the two endpoints must know the direction, the "next hop", to send the traffic in order to get it closer to its destination. This has to be true for every possible combination of source, destination and intermediate node.

This means that any intermediate node (router) has to have some knowledge or understanding of the topology of the network in terms of links and locations of destinations, which is often stored in a **routing table**. The routing tables can also contain other information, such as data about the desirability of a path and certain metrics of the paths. Routers then use some **routing algorithm** to compare these metrics and to translate this knowledge into decisions about preferred paths for given destinations. This decision making process is called **path determination** or **routing** and results in a **forwarding table**. Figure 3-1 shows this process inside three routers. The routers know what interfaces they have and they learn what networks and destinations can be reached through them, using some mechanism. Then they share their routing information, including what they learned from neighbours, with their direct neighbours, often called **peers**. This way, each router can build its own routing tables and construct its own forwarding tables.

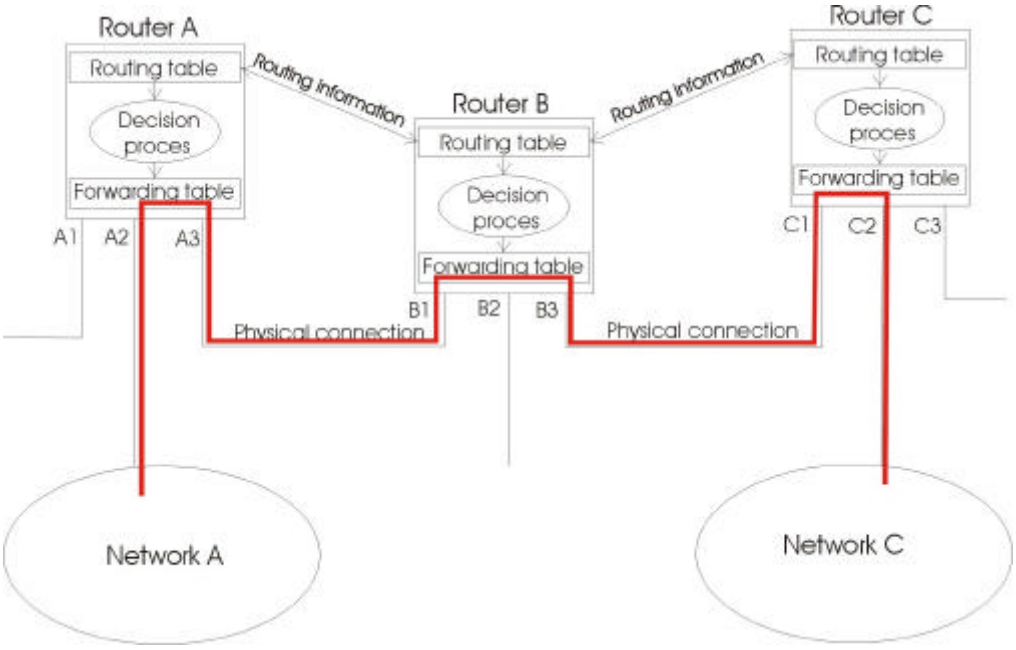


Figure 3-1. Conceptual view of routing and forwarding

Now if a packet arrives for a certain destination, the router will look in its forwarding table and it will copy the packet to the interface that gets the packet closer to its destination. If it doesn't know the interface to copy the packet to, the packet is typically dropped. The process of receiving a packet, doing a lookup and copying the packet is called **forwarding**. Sometimes the term switching is used [ST98]. Figure 3-1 shows an example where a host in network A sends a packet to a host in network C. Each router has three interfaces (1,2 and 3). Assume that host A already knows that the packet is not intended for a host in its own network[3]. It sends the packet to the **default router** (default gateway) for outgoing packets[4], which looks in its forwarding table if it has an entry for network C addresses. It finds that in order to get this packet closer to its destination the best next hop is router B, so it copies the packet on interface A3. Router B and router C, in turn, perform the same action and the packet gets copied to interface B3 and C2 respectively and arrives at the destination network. This forwarding process occurs many times in a short period of time, whenever a packet arrives. The routing process, however, only occurs when a change in the routing information is detected and when a new forwarding table has to be constructed, or after some pre-defined time interval. Note that as a packet moves through the network, its physical layer destination address changes every time to reflect the next hop, but its network layer destination address (in our case the IP address) remains that of the destination host.

There are different kinds of **routing algorithms** and this results in different kinds of routing tables. The routing algorithms initialise, maintain and update the routing tables. Two examples of routing algorithms are **Distance Vector** (DV) and the **Link State** (LS) algorithms. I will give a brief introduction on these two routing algorithms[5],

In DV routing it is required that each router maintains the distance from itself to each possible destination. These distances are computed from distance information (distance vectors) that the routers acquire from their neighbours. These neighbours, in turn, also acquire their distance vectors from their neighbours, and so on. Look at the brief example in Figure 3-2. If router B is a direct neighbour of router A and it knows its distance, 4, than it can send this distance vector to router C. Router C now knows that it can reach router A, and thus network A, through router B and that the total distance is 6. The DV protocol most used in the Internet is RIP or RIP-2[6].

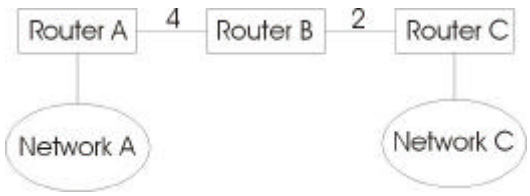


Figure 3-2. Simple example of DV algorithm

LS routing algorithms are based on full knowledge of the entire network topology for each router. Each router first learns about its neighbours and then constructs a link state packet, or LSP, which contains the list of the names of and the cost to each of its neighbours. This LSP is somehow transmitted to every router in the network and the most recent LSP from each other router stored. Now each router can compute the best route to each destination using this complete map of the topology. One of the major advantages of LS over DV is that only LSPs need to be sent and not entire routing tables. Also, if one link state changes only a few LSPs are needed, in DV completely new routing tables need to be constructed. OSPF is the most used LS protocol in the Internet.

At this point we know what a router does with the information in its routing tables, but we still don't know how this information is obtained. Routers communicate with each other and maintain their routing tables by sending and receiving different kinds of messages. By analysing the messages from all other routers, a router can build a detailed picture of the topology of the network. Also, this topology is likely to change in time, which means that the routers continuously have to monitor the network and send each other messages about the current situation. These messages and the information they

[3] How this is done will be described in the next paragraph
[4] In more complex networks with more than one router connected, a protocol like DHCP or router discovery could be used, but most often a static default router is configured at the hosts.
[5] for a complete discussion of DV and LS routing algorithms and their (dis-)advantages look at [TAN96, RP00, HUI95 or STA96].
[6] A note on terminology: When we speak of a DV or LS protocol we mean a routing **protocol** that is based on the DV or LS **algorithm**.

contain are specified in the **routing protocols**. So, routing protocols, are defined by a set of message formats for describing the reachability and preference for network addresses along with rules for processing information learned through these messages [ST98]. Routing protocols are closely related to routing algorithms and together they define how the routing process in a particular (part of the) network is arranged.

Note that this is only one possible source of information, because in some cases the system administrator can provides the router with the necessary information. This fact is stressed here, because it is important to realise that the router needs information and that it can obtain this information from different sources, one of which is from other routers through some routing protocol.

One final remark has to be made here, before we go into the details of routing in the Internet. Most internetworks work with a two level hierarchy where the entire network is divided in **routing domains**. A routing domain is generally considered to be a portion of an internetwork under common administrative authority that is regulated by a particular set of administrative guidelines. In this context we can distinguish between two types of routers. One set of routers can only communicate with other routers within the same routing domain and other routers can communicate both within and between routing domains. A routing domain in the Internet is often called an **autonomous system** [ITO99]. The protocols running within an autonomous system are called **Internal Gateway Protocols** (IGPs) and those running between autonomous systems are called **External Gateway Protocols** (EGPs). This will be described in more detail later in this chapter, first the we will look at basic routing in the Internet.

IP addressing and routing

In the previous chapter we have already seen the syntax of IP addresses, with a network prefix and a host number. In order to support different sized networks, originally different address classes were designed. These classes fix the boundary between the network prefix and the host number at different points within the 32-bit address. This is often referred to as “classful” addressing. Because original routing protocols did not work with “masks” to identify the length of the network prefix, each address contains a self-encoding key that identifies the dividing point between the network prefix and the host address. If an IP address starts with 110, for example, the dividing point is between the 23rd and 24th bit. The different classes and some of their properties are given in Table 3-1[TAN96, CS96].

The routers, however, do use a mask to determine the network prefix. This mask consists of ones as the most significant bits and zeros as least significant bits. The length of the network prefix determines the number of one’s. A class B network, for example has the mask 11111111.11111111.00000000.00000000 or in dotted-decimal notation 255.255.0.0. Now, when a packet arrived at a router, the router would look at the first bits of the destination address and determine from those, the class and thus the length of the network prefix. It then extracted the network-prefix from the destination address by doing a bitwise AND with the mask. The remaining network-prefix was then looked up in the forwarding tables to determine the next hop in order to forward the packet closer to the associated destination network.

Class	Address Format (n = network, h = host)	Range of host addresses	Networks[7]	Hosts[8]
A	0nnnnnnn.hhhhhhhh.hhhhhhhh.hhhhhhhh	1.0.0.0 to 127.255.255.255	126 (2^7-2)	16777214 (2^{24}-2)
B	10nnnnnn.nnnnnnnn.hhhhhhhh.hhhhhhhh	128.0.0.0 to 191.255.255.255	16384 (2^{14})	65534 (2^{16}-2)
C	110nnnnn.nnnnnnnn.nnnnnnnn.hhhhhhhh	192.0.0.0 to 223.255.255.255	2097152 (2^{21})	254 (2^8-2)

Table 3-1. Properties of the three most common classes

If we look at a class A network, for example, we see that it has an 8-bit network-prefix with the highest order bit set to 0 and a seven-bit network number, followed by a 24-bit host-number. Instead of referring to a class A network, we also speak of a “/8” (slash eight) network, because it has an 8-bit network-prefix [CS96]. A class B network is referred to as a “/16” network and a class C network as a “/24” network. Next to these three classes there are two additional classes. Class D addresses, start

[7] 2 has to be subtracted from 2^7 because the /8 networks 0.0.0.0 (default route) and 127.0.0.0 (loopback) are special and reserved.
[8] The host calculation requires that 2 is subtracted because the all-0s (“this network”) and the all-1s(“broadcast”) host-numbers may not be assigned to individual hosts.

with 1-1-1-0 as their most significant bits and are used to support IP multicasting and Class E addresses start with 1-1-1-1 and are reserved for experimental use.

Subnetting

In 1985, the technique of subnetting was introduced to overcome some problems that occurred. Examples hereof are growing routing tables and network engineers having to request for a new network number, before any extra networks could be installed within a particular organisation, which also led to inefficient usage of the IPv4 addresses. By adding another level of hierarchy to the addressing structure only one or a few network numbers had to be assigned to each organisation.

In this new situation the organisation assigned a different subnetwork number for each of its internal networks. The internal subnet structure of a network is never visible outside of the organisation's private network; only the network-prefix is announced, so only one routing table entry is needed in routers outside the subnetted environment. Internet routers use only the network-prefix of the destination address to route traffic to a subnetted environment. Routers within the subnetted environment use the extended-network-prefix to route traffic between individual subnets, as shown in Figure 3-3 [CS96]. Note that the term subnet here refers to the addressing of portions of a network and not to a physical network or the communications subnet as mentioned in Chapter 2.

This extended-network-prefix was originally identified by the subnetmask, which has its bits set to 1 if the bits of the corresponding IP address are meant to be treated as part of the extended-network-prefix, and set to 0 if the system should treat the bit as part of the host-number. For example if the first 27 bits of an address should be treated as the extended-network prefix, the subnetmask is 11111111.11111111.11111111.11100000, or in dotted-decimal notation 255.255.255.224. However, most standards describing modern routing protocols refer to the extended-network-prefix length rather than the subnet mask, in which case the notation becomes "/27". Still, the routing protocols today carry the subnetmask, instead of, for example, a one-byte field in their header that contains the extended-network-prefix length.

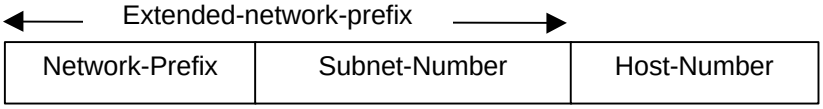


Figure 3-3. Subnetting and the extended-network-prefix

Later, in RFC 1009, it was specified how a subnetted network could use more than one subnetmask. When an IP network is assigned more than one subnet mask, it is considered a network with "variable length subnet masks (VLSM)" since the extended network-prefixes have different lengths. One of the two major advantages of this approach is that there can be a more efficient use of an organisation's assigned IP address space, and thus of the total IPv4 address space. The second advantage is that multiple subnet masks permit route-aggregation which can significantly reduce the amount of routing information at the "top" level within an organisation's routing domain. At this point it is important to realise that this is only of concern for the interior gateway or routing protocol used within the organisation, the exterior world will still only "see" the network prefix. If an organisation wishes to deploy VLSM in a complex topology OSPF or I-IS-IS or RIP-2 are required as interior gateway protocol (IGP). Because this study mainly focuses on inter-domain routing this subject will not be explored here further.

CIDR

However, subnetting and VLSM were still not adequate too solve the problems the Internet faced at that time. Many organisations requested a class B subnet, because this left them with enough room to organise their subnetting. Since there are only 16383 of such networks available, this part of the address space was "consumed" very fast [HUI95]. The alternative of serving these organisations with several class C addresses, would have led to a major increase of the size of the routing tables. Because for every class C network an extra entry in the routing table would be needed. The Classless

Inter-Domain Routing (CIDR) was designed as a solution to this problem. The two main features of CIDR are:

- CIDR eliminates the traditional concept of Class A, B or C networks, thus allowing for a more efficient allocation of the Ipv4 address space.
- CIDR support route aggregation where a single routing table entry can represent the address space of perhaps thousands of traditional classful routes. Route aggregation helps control the amount of routing information in the routing tables.

There are three basic requirements for CIDR (as well as for VLSM) to be successfully deployed:
1. The routing protocols, both interior and exterior, must carry network-prefix information with each route advertisement. (extended-network-prefix for VLSM)
2. All routers must implement a consistent forwarding algorithm based on the "**longest match**."
3. For route aggregation to occur, addresses must be assigned so that they have topological significance.

The first requirement is met by most modern routing protocols, such as OSPF, RIP-2 and I-IS-IS. They provide the (extended) network prefix or mask value with each route advertisement.

The second requirement is interesting in our explanation of routing in the Internet. A route with a longer extended-network-prefix describes a smaller set of destinations than the same route with a shorter extended-network-prefix. This means that a route with a longer extended-network-prefix is said to be "**more specific**" while a route with a shorter extended-network-prefix is said to be "**less specific**". The longest match algorithm makes sure that routers use the route with the longest matching extended-network-prefix (most specific matching route) when forwarding traffic [CS96].

Too see this, let us look at a small example, borrowed from [CS96]. An organisation has the IP address of 200.25.17.25 and it is a customer of an ISP 1 that owns the IP block 200.25.0.0/16. Using CIDR, this ISP will only announce this prefix to the rest of the Internet, instead of announcing individual routes for every IP address or subnetwork. Now if a router hears this announcement it knows that traffic for any address that falls within the 200.25.0.0/16, such as 200.25.17.25 block has to be sent to a specific router. So far, so good.

Now, if the customer decides to move to ISP 2 who owns the IP block 199.30.0.0/16, but if it keeps its IP address of 200.25.17.25, ISP 2 has to advertise this IP address as an "exception". So, along with its standard 199.30.0.0/16 announcement it will send an announcement of 200.25.17.25 to the neighbours. Now if a router wishes to send traffic to 200.25.17.25 it will find a route for 200.25.0.0/16 which is valid, but also one for 200.25.17.25/32, which is more specific, because more bits (in fact, all bits) of this route correspond to the requested destination address. Therefore the router must choose this more specific match and send the traffic to the appropriate router, closer to ISP 2. Note that these exceptions are undesirable in the light of routing table size reduction, however they are needed to ensure full connectivity.

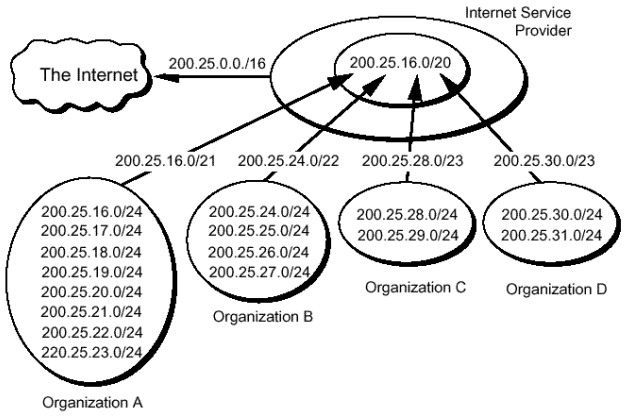


Figure 3-4. CIDR reduces the size of Internet Routing Tables [CS96]

The third requirement is needed to make sure the growth of the routing tables can really be controlled. In the next Figure, if addresses that cannot be aggregated to a single advertisement are assigned within, for example organisation A, more than one prefix has to be advertised to the Internet. This also has implications for situations where organisations change between ISP's. For the organisation it is best if it can keep its address space and if the new ISP would advertise an "exception" (more specific) route into the Internet. However, for the effectiveness of CIDR it would be best if the organisations would be assigned a new block of the new ISP's address space [CS96].

Splitting up the Internet in Autonomous Systems

As we have seen in the previous chapter, the Internet grew out of the Arpanet and in the beginning there was only a single network. All routers (gateways), shared routing information through the same gateway-to-gateway protocol (GGP). The routing tables included entries and metrics for all the IP networks in the Internet [HUI95]. Along with the ever increasing routing overhead as more and more networks got connected, the management of this very large network caused another problem. As the number of routers increased, so did the number of types of routers, all using their own specific implementation of GGP [HUI95]. This made it very difficult to maintain the network and isolate faults. Also the deployment of new routing algorithms caused enormous problems. This called for a different approach.

It was decided that the Internet would be split up into a set of autonomous systems and that a two level hierarchy of routing protocols would be used. This was already introduced at the beginning of this chapter. Some **internal gateway protocol** (**IGP**) would be used to exchange routing information between the routers within an autonomous system. The primary function of an IGP is to determine the most optimal path between any two nodes within a given network. The term optimal refers to some sort of **cost** of the path, which is determined by looking at one or more **metrics** associated for each link in the network. A metric is typically the delay, so the cost of a path is the sum of all delays of the links on that path. Examples of IGPs used in the Internet today are RIP (Routing Information Protocol) and OSPF. Note that the area of network infrastructure (mostly routers) over which an IGP runs defines the boundary of an autonomous system [ST98]. However, it is possible to run more than one IGP in an autonomous system, therefore this is not always true. A more exact definition of autonomous systems in the Internet is given in the next paragraph.

In order to exchange routing information between routers located in different autonomous systems an **external gateway protocol** (**EGP**) is used. This exchange of routing information is necessary to allow for the forwarding of traffic across the borders of an autonomous system. Most EGPs include some mechanisms to allow for **policy routing**, which means that there is some form of control over which routing information crosses the border between two autonomous systems. The current and single de facto EGP in the Internet is BGP4. See Figure 3-5 for a simple topology of autonomous systems and the scope of the IGP and EGP. Note that for the external routers R1, R2 and R2 the world consists of the other external routers and their own internal peers. The internal routers only "see" their internal peers and the external router of their own autonomous system, with which they "speak" IGP.

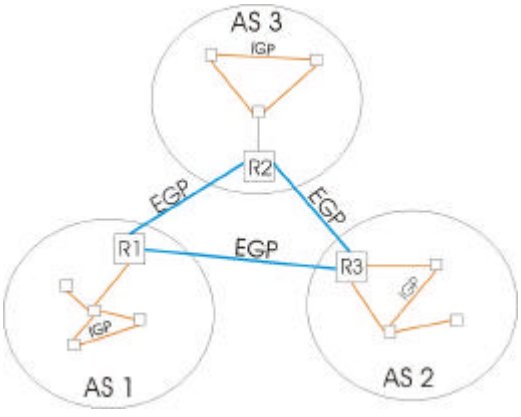


Figure 3-5. A simple example of autonomous systems topology

With this approach different autonomous systems can run completely different IGPs, which makes updating, managing and fault-isolation more straightforward and less risky in the light of global connectivity between autonomous systems. Alternative terms for IGP and EGP are intra-domain routing protocol and inter-domain routing protocol, respectively.

To summarise using a sentence from [ST98]: “Interdomain routing protocols such as BGP are the glue that ties the various networks together to make sure the user of one network can make use of the resource no matter where it connects to the network”.

The previous description points out that the usage of autonomous systems was an explicit design choice to make the updating and troubleshooting of the routing protocols more manageable. This means that the border of an autonomous system is only relevant in the context of routing and routing protocols and that it doesn't imply any organisational border. For example, the autonomous system 1103 (SURFNET)[9] consists of networks, such as those of the Universities of Delft (DUNET), Groningen (RUGNET) and Eindhoven (TUENET), but also of organisations such as KPN Research (PTT-RESEARCH, TNO (TNO-NET) and the Dutch ministry of VROM (MIN-VROM). Surfnet is developed and maintained by one organisation, Surfnet B.V. and has about 200 research and higher education institutions connected. The topology of the Surfnet backbone is shown in Figure 3-6 and we see from this picture that an autonomous system is not per definition centred at on geographical location.

Figure 3-6. The Surfnet (AS1103) topology [Source: http://www.surfnet.nl/en/surfnet-organisation/]

Autonomous Systems

The previous paragraph introduced the concept of autonomous systems. In this paragraph we try to get a good understanding of what the term autonomous system exactly stands for. This has proven to be not so straightforward, because in most literature an autonomous system is described as a part of an internet that runs a single IGP, but then if we look at the definition of an IGP it is defined as a routing protocol that runs within a single autonomous system. We need to obtain a definition that doesn’t follow this circular approach. There are many different, more or less overlapping, definitions of the term in the literature. I will give some examples:

[9] The capitalised name between brackets is the description of a route object as it appears in the RIPE database, which will be explained in a later chapter.

> *An autonomous system in the Internet refers to a portion of the network, usually within the control of one organisation and usually running a single routing protocol. Routing between Autonomous Systems is done with a protocol that is defined as being an interdomain or inter-AS protocol or an exterior gateway protocol. The term autonomous system within the IP community is synonymous with the term routing domain in the ISO community [RP00].*
>
> *An AS is a collection of routers operated in a co-ordinated way so that the routers implement the same routing policy; typically operated by a single administrative entity [ST98].*
>
> *On the Internet, an autonomous system (AS) is the unit of router policy, either a single network or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of a single administrative entity (such as a university, a business enterprise, or a business division). An autonomous system is assigned a globally unique number, sometimes called an Autonomous System Number (ASN). [WHATIS]*

All these definitions provide some valuable insight in the meaning and use of autonomous systems, but probably the most complete definition is in [RFC 1771]:

> *The classic definition of an Autonomous System is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets to other ASs. Since this classic definition was developed, it has become common for a single AS to use several interior gateway protocols and sometimes several sets of metrics within an AS. The use of the term Autonomous System here stresses the fact that, even when multiple IGPs and metrics are used, the administration of an AS appears to other ASs to have a single coherent interior routing plan and presents a consistent picture of what destinations are reachable through it.*

This last definition points out an important aspect of autonomous systems, which helps us in understanding how interdomain routing in the modern Internet works. The last part of this definition shows that the internal structure of an autonomous system is shielded off from the other autonomous systems. By some mechanism and according to its routing policy an AS announces which destinations can be reached through that AS. Other ASs do not have to worry about the internal organisation of the first AS, they just deliver their packets to a "**Border Router**" or "Border Gateway" of that specific AS and leave it up to that router to route the packets closer to its destination. How this will be discussed works in more detail later in this chapter. Also the meaning of the term routing policy will become clear later on.

A second aspect that is of importance is that an autonomous system is a collection of routers. These routers are mostly part of the one and the same IP prefix, but often more IP prefixes reside under one autonomous system. Therefore two routers with totally different IP addresses could be part of the same autonomous system.

These two aspects, which are the most important aspects of autonomous systems to our research, probably makes the definition of autonomous systems in [RFC1930] the most suitable for us:

> *An autonomous system is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy.*

Note that traffic arriving at a border router of an autonomous system, intended for a network contained in that autonomous system, in principle does not need to pass any other autonomous system to get to its destination.

At this point the only term that is left to be explained for a full understanding of autonomous systems is routing policy. In short, a **routing policy** determines what routing information will pass the border of an autonomous system, and this in turn determines which traffic may pass the borders of an autonomous system. At the end of this chapter, after a description of BGP4, the subject of routing policy will be briefly addresses to.

BGP4

The Border Gateway Protocol (BGP) is an inter-Autonomous System routing protocol. Its main purpose is to allow BGP speaking systems located at neighbouring autonomous systems to exchange network reachability information with each other and thus to exchange this reachability information between autonomous systems (ASs). The information that is being exchanged consists of network prefixes that can be reached, including information on the list of autonomous systems that the information has traversed. This list of ASs, known as the AS_PATH, allows receivers to construct a graph of AS connectivity. This way routing loops can be avoided and some policy decisions at the AS level may be enforced. BGP is specified in [RFC1771].

This concept of using the AS_PATH in BGP is sometimes referred to as **path vector** routing, as an enhancement of distance vector routing. The distance vector approach did not give enough protection against routing loops, while “using a link state approach was deemed unrealistic”. The drawback is that because of the need to exchange the full AS_PATH, the routing messages will be bigger, and also the amount of memory needed in routers increases [HUI95].

The current version of BGP is BGP4, which also is the current de facto standard for inter-domain routing in internetworks based on IP and thus in the Internet. BGP4 has some mechanisms for supporting classless interdomain routing (CIDR), which include support for advertising an IP prefix along with its length (or mask), instead of using the concept of network "class". Also aggregation of routes, including AS paths is introduced in this version of BGP. Previous external gateway protocols used in the Internet are BGP-3 and EGP.

First we look at some topological considerations that are important for a good understanding of BGP and interdomain routing in the Internet.

Topological model

Since this study focuses on the topology of the Internet at the level of autonomous systems, it is important to consider the topological implications of BGP and Autonomous Systems. Here some special considerations and terminology related to ASs and BGP are presented, before going in to details about how BGP works.

The first thing to explore is the question when a connection between two autonomous systems exists. [RFC1772] speaks of a connection between two ASs if there is a physical connection and a BGP connection. The physical connection refers to a shared Data Link subnetwork at which each AS has at least one border gateway or border router. This subnetwork can be anything from an Ethernet to Token Ring to a direct connection. What's important is that these (at least) two routers can forward packets to each other without resorting to any form of routing. The BGP connection means that there is a BGP session[10] between BGP speakers in each of the ASs, and this session communicates those routes that can be used for specific destinations via the advertising AS. A **BGP Speaker** is a host or router that can exchange BGP messages with other BGP speaking systems.

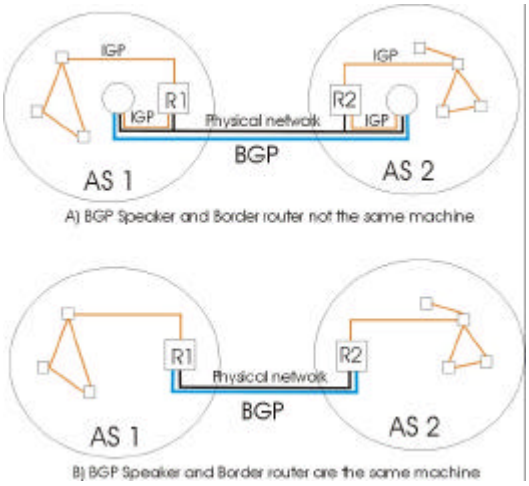


Figure 3-7. Possible connections between ASs

Note that the specification doesn’t require that the BGP speaker itself is a border router, as in Figure 3-7A, which would mean that by some mechanism the border router should get routing

[10] The meaning of a BGP session will be explained a bit further on in this paragraph

information from the BGP speaker, for example using some IGP. Remember that the actual transportation of packets, or the forwarding, works with IP prefixes, and the way the border routers learn about these prefixes is not important for this forwarding function. However, most often these two functions are combined in the border router(s), and I will use the term **BGP router** in this general case, see Figure 3-7B.

Although it is not obligatory, an additional constraint, mentioned in [RFC1772], on the BGP speakers is that they also share the same Data Link subnetwork with their border routers and with their BGP neighbours. However, since most often the border router and the BGP speaker are the same machine, this is already the case. Still, since BGP uses TCP as its transport protocol, it could be possible for two BGP speakers to exchange BGP messages, without residing on the same physical network. This however is only allowed for I-BGP, which we will look at later. The most common situation is where the border router is also the BGP speaker and where two neighbouring BGP routers are directly connected.

Because BGP uses TCP as its transport protocol, it works between exactly two nodes in what is called a **BGP session**. There may be many BGP sessions at any given time in a network and a BGP speaker may be participating in many BGP sessions, but it is important to realise that for one BGP session only two BGP speakers are involved. These two routers are referred to as **BGP peers**.

Some notes on terminology have to be made here. Connections between BGP speakers of different autonomous systems are referred to as **external links**, however also connections with BGP speakers within the same autonomous system are possible. These connections are called **internal links**. Following this same terminology, a peer in a different AS is called an **external peer** and a peer in the same AS is called an **internal peer**. Sometimes the terms internal BGP (I-BGP) and external BGP (E-BGP) are used, which provides us with useful acronyms, see Figure 3-8.

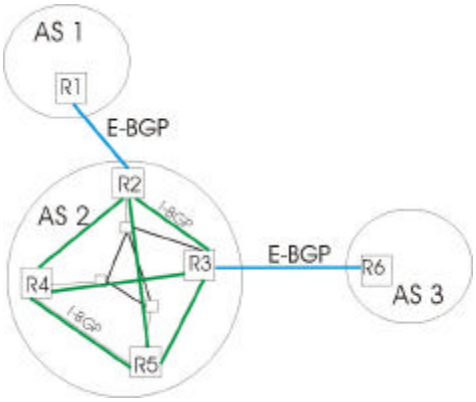


Figure 3-8. I-BGP and E-BGP

In order to understand another aspect of the topology of ASs, the concept of **transit traffic** needs to be introduced here. Much of the traffic that is carried within an AS either originates or terminates at that AS. In practise this means that either the source or destination IP address is part of that AS. This traffic is called **local traffic**. All other traffic that doesn't fit this description is called transit traffic. One of the major goals of BGP is to control the flow of transit traffic [RFC1772].

Now we can distinguish three different types of autonomous systems, based on how they deal with transit traffic:

- A **stub AS** is an AS that has only a single connection to one other AS, and thus such an AS can only carry local traffic.
- A **multihomed AS** has connections to more than one other AS, but refuses to carry transit traffic.
- A **transit AS** has connections to more than one other AS and is designed to carry both transit and local traffic.

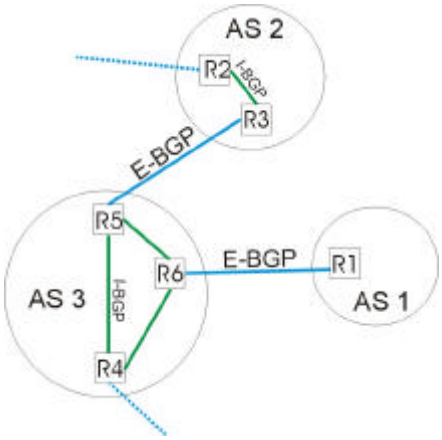


Figure 3-9. Stub, multihomed and transit AS

Especially transit ASs, but also multihomed ASs are the subject of routing policies, because they have to accept or reject certain kinds of traffic, based on political, security or economic considerations. See Figure 3-9, where AS1 is a stub-AS, AS2 could be a

multihomed AS, if its routing policy doesn't accept traffic destined for a AS1 or AS3 (otherwise it is also a transit AS), and AS3 is a transit AS, since it provides connectivity for AS1.

As we can see, multihomed and transit ASs have multiple BGP sessions, although they not necessarily have multiple BGP routers, because one BGP router can maintain BGP sessions with multiple external peers.

Since the AS needs to represent itself to the other ASs as having a single and clearly defined routing policy, it is important that the internal BGP routers can maintain a consistent view of the routes exterior to the AS and that they have a consistent view to represent to the outside world of the networks that reside inside the AS. Using a common set of policies, the BGP routers arrive at an agreement as to which border routers will serve as exit/entry points for particular destinations outside or destinations within the AS. This means that the internal BGP routers need to be able to communicate with each other. When using BGP for this communication, again, we speak of I-BGP or an internal link. Because only two BGP routers can participate in a BGP session, all BGP routers within an AS must be fully meshed **logically** [ITO99]. This doesn't mean they have to be fully meshed at the physical level. As long as there is an IGP running that allows the two neighbours to reach one another, I-BGP peers do not have to be directly connected. For an example of this, look back at Figure 3-8 inside AS2. The underlying physical connections provide for the necessary logical connectivity. Note that this only holds for I-BGP, since the two routers can find each other because of some IGP. With E-BGP however, the two communicating routers need to be on the same physical network, as explained earlier.

Conceptual model of BGP operation

As we've seen, when two BGP peers exchange routing information dynamically with BGP, they first establish a TCP connection between them and then pass BGP messages over that connection. The BGP router uses this information to update its own routing and forwarding tables. Among the messages they exchange are messages to inform the neighbour about new routes that are active, or about old routes that are no longer active [ST98]. This process and the messages are described in the remainder of this chapter.

Before an AS exchanges any information with an external AS, BGP ensures that the networks within the AS are reachable. This is done by I-BGP messages among BGP routers within the AS and by redistributing BGP routing information to IGPs that run within the AS, for the internal routers. A router that implements BGP will also implement an IGP. Through this IGP or through static information provided by a system administrator, the BGP router can build up a picture of the internal topology and construct a routing table [STA96]. This way the AS can provide external neighbours with a consistent view of the networks that reside (originate) in the AS. The interaction of BGP with any of the IGPs (or with other EGPs) could be subject of extensive study, but due to time limitations, this has to fall outside the scope of this report.

Also before any communication can begin, the BGP routers have to know of each others existence. This is manually configured at each of the neighbouring BGP routers. Each BGP router keeps a **BGP neighbour table** specifying that a peer identified by a certain IP address, belongs to a specified AS. As was said before, since most often the BGP peers are directly connected, the IP addresses in this table are usually the IP addresses of the interface at the other end of the connections [CIS01].Now BGP sessions can be established.

First the peers send messages to open and confirm the connection parameters and to identify themselves to each other. This exchange of messages is important to BGP, but not of real interest to our research, and therefore I will not go into this subject further. After this connection has been established, the peers can be assured that the messages will be delivered reliably by TCP, which means that the BGP messages themselves can be very simple and with minimum overhead. If the connection is refused by one of the peers, or if some error occurs, some notification will be sent and the TCP connection will be closed. Periodically KeepAlive messages are sent to ensure the connection is still up.

When the 2 endpoints accepted the connection, the BGP session is fully active and now routes can be exchanged. Each of the endpoints will advertise a route for every prefix that it wants the other end to know, which means that the initial data flow is the entire BGP routing table intended for the other end. This, in turn, means that a large number of initial messages need to be sent. However, when a router announces a prefix to one of its BGP neighbours, this information is considered valid until the first router explicitly advertises that the information is no longer valid, or until the BGP session itself is lost. The routing information does not need to be refreshed, only updated. After the TCP connection is broken, each end must stop using the routing information that it heard from the other [ST98].

Along with an advertised prefix, a number of attributes are sent, associated with that prefix. These attributes are used heavily in BGP to carry a wide range of information. [ST98]. These attributes are discussed later on!

So how does a BGP speaking system deal with the routing information in the received messages? To understand this, the term route needs to be introduced and explained. A **route** is defined as a unit of information that pairs a destination with the attributes of a path to that destination [RFC1771]. Basically, this is the unit of information that is most important to BGP and it contains all the information concerning a certain destination (some network prefix) that is of importance for the routing process. The network prefix, the AS-path and its attributes are reported in the UPDATE message, which will be addressed to later in this chapter.

These routes are stored in the Routing Information Bases (RIBs), namely, the adj-RIBs-In[11], the Loc-RIB and the adj-RIBs-Out. The adj-RIBs-In contain unprocessed routing information that the local BGP-speaker has received from its peers. The Loc-RIB contains the routes that have been selected by the local speaker, by applying its local policies to the information in the adj-RIBs-In.
The adj-RIBs-Out store the information that has been selected for advertisement to its peers, by applying policies to the information in the Loc-RIB. Before advertising routes in the adj-RIBs-Out to its peers, using an UPDATE message, the local BGP speaker may modify or add to the path attributes of the route.

There is one adj-RIB-In and one adj-RIB-Out for every BGP peer, and there is exactly one Loc-RIB. Note that these tables need not be implemented as separate tables, this is only the conceptual model. Finally there is a forwarding information base which contains the next hop for every route. This table is used for the speedy forwarding of the packets and the information in this table is derived by some decision process from the Loc-Rib.

The above description shows that upon receiving a prefix from its peer, a BGP speaker does not necessarily have to accept it. It can selectively accept and reject routes based on its policy. Also multiple Adj-RIBs-In can contain different paths to the same network prefix, and thus a selection has to be made. The process of filling the RIBs, analysing the information and applying policies to the routes and finally constructing the forwarding information base can be referred to as routing. Now when a packet for a certain destination arrives at the BGP speaking router, the next hop is looked-up in the forwarding base and the packet is copied closer to its destination. This process is called forwarding, as we have already seen earlier in this chapter.

[11] the term “adj” in adj-RIB-In and in adj-RIB-Out stands for adjacent.

Figure 3-10 summarises the BGP routing information and storage as described here. Note that the red ring represent the area of routing policy; information entering or exiting this conceptual “ring” is subject to change according to the routing policies configured at the specific BGP router.

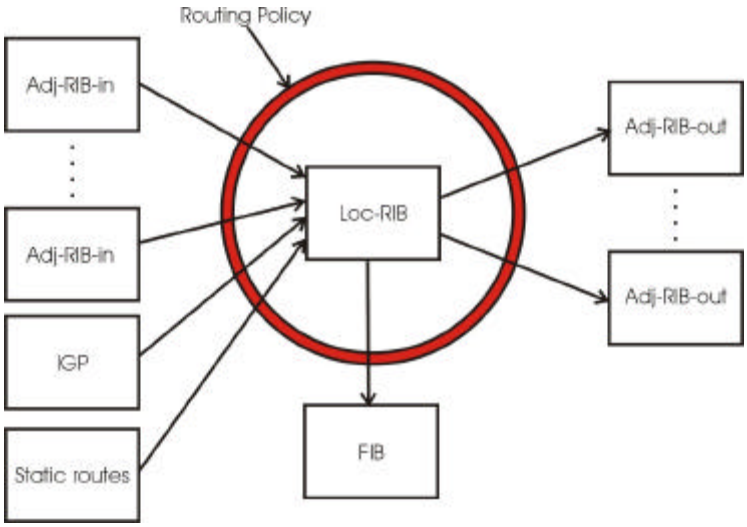


Figure 3-10. Routing information and routing policy

Next to mechanisms to advertise new routes, three mechanisms to withdraw old routes are specified. First an IP prefix that reflects destinations that were previously advertised to a specific BGP speaker can be advertised in the WITHDRAWN routes field of the UPDATE message. Second a replacement route with the same destination can be advertised, or third, the connection between the two BGP speakers can be closed, which implicitly removes all routes that the two peers had previously exchanged.

The Messages

There are four messages that are used in BGP4. These messages have a common header and the type field in this header specifies the BGP message being sent. The possible types are:

1. OPEN
2. UPDATE
3. NOTIFICATION
4. KEEPALIVE

For details about these messages and of the header refer to [RFC1771 and ST98], here I will only briefly describe the function of the different messages, and the focus will be on the UPDATE messages, which contain the routing information.

The OPEN message is used for identification and for protocol parameters negotiation. The "My Autonomous System"-field in the OPEN message specifies the Autonomous System Number of the sender, along with the "BGP identifier"-field , which typically holds the IP address of the BGP speaker, this is used for identification. Also the length of the hold timer is part of this message. The hold timer is the maximum length of time that one endpoint will wait to hear something from the other endpoint, before assuming that the BGP session is down.

The NOTIFICATION message is sent whenever an error condition is detected. The BGP connection is closed immediately after sending it. KEEPALIVE messages are exchanged between peers often enough as not to cause the hold timer to expire.

The UPDATE message is the primary message used to communicate routing information between two BGP peers [ST98]. An UPDATE message is used to advertise a single feasible route to a peer, or to withdraw multiple unfeasible routes from service. This may occur simultaneously in one message.

Along with the standard BGP header, the UPDATE message can optionally contain the following fields:

- Unfeasible Routes Length
- Withdrawn Routes
- Total Path Attribute Length
- Path Attributes
- Network Layer Reachability Information (NLRI)

The Unfeasible Routes Length Field indicates the length of the Withdrawn Routes Field in octets. The Withdrawn Routes Field contains a list of IP prefixes for which the sender of the UPDATE message no longer wishes to forward packets. By sending an UPDATE message with withdrawn routes for certain prefixes, the sender wishes to prevent the receiver to send traffic destined for the prefixes through the senders autonomous system. Each IP address prefix in the Withdrawn Routes Field is encoded as a 2-tuple of the form <length, prefix>, where the length field indicates the length of the prefix (the mask), and the prefix field contains the prefix itself.

The Total Path Attribute Length indicates the length of the Path Attributes Field, which contains a list of BGP attributes associated with the prefixes in the Network Layer Reachability Field. These attributes are an important feature of BGP and they describe the prefixes in ways such as how the prefix came to be routed by BGP, which path of ASs the advertisement has travelled so far and metrics expressing degrees of preference for this prefix [ST98]. So, the attributes contain important information, also used by the routing process in a BGP speaking router.

Each attribute in the Path Attributes Field is encoded as attribute type, attribute length and attribute value. The attribute type field in turn is a two-octet field where the first octet defines the Attribute Flags and the second octet the Attribute Type Code. Most bits of the Attribute Flag field are not important to us, but the values of the first three leading bits in the Attribute Flag field define in which category the attribute fits:

1. Well-known mandatory attributes must be recognised by every BGP implementation and included in every UPDATE message and passed along to other BGP peers upon reception and after proper updating.
2. Well-known discretionary attributes must be recognised by every BGP implementation, needn't be included in every UPDATE message, but must be passed along to other BGP peers.
3. Optional Transitive attributes need not be recognised by every BGP implementation, but they must be passed along to other BGP peers.
4. Optional non-transitive attributes needn't be recognised by every BGP implementation, and they must be quietly ignored and not passed along to other BGP peers.

The attribute Type Code field specifies the type of attribute, the attribute length field the length and the attribute value contains the actual value of the specified attribute and is parsed and processed according to the Attribute Flags and the Attribute Type Field. Additional attributes have been added, but only the base path attributes are described in the next paragraph.

Finally the Network Layer Reachability Field contains a list of IP address prefixes, whose length can be calculated from the other length fields and the length of the total message and of the standard header. Each IP address prefix is again encoded as a 2-tuple of the form <length, prefix> as described earlier.

All the attributes in the Path Attributes field apply to all the prefixes in the NLRI field. Therefore, an UPDATE message can only advertise one route. Although multiple prefixes can be announced in one NLRI field, these prefixes and the associated Path Attributes are referred to as one route by the specification in [RFC1771]. This happens when a the prefixes can't be aggregated to one prefix, but when the attributes are all the same, which means that the prefixes reside in the same AS.

The Path Attributes

As we have seen, in every UPDATE message a variable length sequence of path attributes is present. These Attributes are among the most important features of BGP4 and they contain information about the prefixes in the Network Layer Reachability Information (NLRI) field. Along with the prefixes, these attributes contain the important information. The base specification includes a number of attributes, but additional attributes have been added over time and are commonly seen in the Internet today [ST98].

ORIGIN

This is a well-known mandatory attribute that defines the origin of the path information. This attribute describes how a prefix came to be routed by BGP at the origin AS, thus this attribute is generated by the AS that originates the associated routing information. Prefixes can be learned from directly connected interfaces, from manually configured static routes, through dynamic internal routing protocols (IGPs) or through dynamic external routing protocols (EGPs). The administrator of a router can configure it to take prefixes learned from such sources and then route them through BGP. Possible values for the ORIGIN attribute are IGP, EGP or INCOMPLETE, of which the latter means that the information came from some source other that the IGP or EGP. This is mostly seen at static routers.

AS_PATH

This well-known, mandatory attribute contains the autonomous systems through which the announcement for the prefix has passed. As a prefix is passed on between ASs, each AS adds its ASN to the AS-PATH attribute. Lets look at the example in the Figure 3-11.

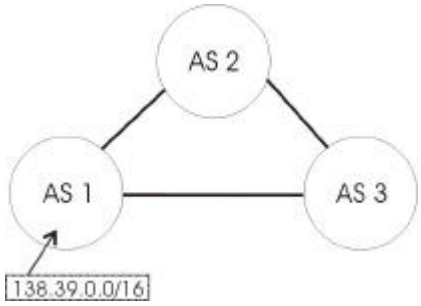


Figure 3-11. AS_PATH example [ST98]

If a BGP speaker in AS 1 announces a prefix 138.39.0.0/16 to AS 2 it places its own AS number in the AS_PATH. This way AS 2 can see that AS 1 originates the prefix, because it is the first and only ASN in the AS_PATH attribute. In this context, AS 1 is called the origin AS. If now AS 2 chooses to advertise this route to AS 3 it adds its own AS number to the AS_PATH and sends the UPDATE to AS 3. Now AS 3 knows that the route originates in AS 1 and that the information was passed to AS 2 and then to AS 3.

This is useful information in relation to policy routing, which we will look at later. However, this information is also valuable in detecting looping announcements. If AS 3 chooses to advertise the route for 138.39.0.0/16 learned from AS 2 to AS 1 it is possible for AS 1 to reject the route, because it knows that it actually originated in AS 1.

However, some implementations support configuration to accept prefixes with the local ASN in the AS_PATH. This could be useful for what is called *partition healing* [ST98], where an AS becomes partitioned and prefixes in one part of the AS are unreachable directly from the other part of the AS. If in the example of Figure 3-11, for some reason AS1 got divided this would mean that packets originating at the first part and destined to the other part need to traverse AS 3 and AS2. In this case it would be helpful if the route announcement could go from the first part of AS1 to the second part of AS1. Most often this is not used.

The AS-PATH attribute is composed of a sequence of AS path segments, each of which has a type, a length and a value. The path segment type can be either AS_SET, which is an unordered set of ASs a route in the UPDATE message has traversed, or AS_SEQUENCE, which is an ordered set of ASs a route in the UPDATE message has traversed. The length field contains the number of ASs in the path segment value field. [RFC1771]. In practise, most routes advertised in BGP have a single AS_SEQUENCE in the AS_PATH attribute. The purpose of AS_SET is to support aggregation of

routes with different AS_PATHs. However, most often aggregation is done on more-specific prefixes that have identical AS_PATH attributes.
When a BGP speaker A advertises a route to BGP speaker B which it has learned from BGP speaker C's UPDATE message, it will modify the route's AS_PATH in a certain way, depending on whether the BGP speaker B is an internal or an external peer. If speaker B is an internal peer, meaning it is located within the same autonomous system, the AS_PATH attribute shall not be modified. However if speaker B is an external peer, speaker A will modify the AS_PATH attribute. It first looks at the first path segment of the AS_PATH, and if that is of the type AS_SEQUENCE, it simply puts its own AS number in the sequence as the last element (leftmost position). If, however first element of the existing AS_PATH attribute is of type AS_SET, speaker A shall prepend a new path segment of type AS_SEQUENCE to the AS_PATH, including its own AS number in that segment.

NEXT_HOP

The NEXT_HOP path attribute defines the IP address of the border router that should be used as the next hop to the destinations listed in the UPDATE message. Often the NEXT_HOP address is the same as the BGP speaker that sends the UPDATE, but this is not necessarily true [ST98].
The specification talks about the next hops that are allowed. The most important rule is that any internal (to the advertising BGP router) border router that shares a common subnet with both the local (advertising) and remote (receiving) BGP router can be advertised as next hop. This is important, for example, if certain parts of an AS can only be reached through a non-BGP speaking router. This router, however has to be on the same subnetwork as the BGP router in the remote AS. Remember, when the term BGP router is used. This refers to a BGP *speaking* router, which is the most common case, although “non-routing” BGP speakers and “non-BGP speaking” border routers also exist.

MULTI_EXIT_DISCRIMINATOR (MED)

This is an optional non-transitive attribute whose value may be used by the BGP speakers decision process to discriminate among multiple exit points to a neighbouring autonomous system. This implies that between two ASs there can be multiple links to choose from. If this is the case, the MED field can be used to carry a metric expressing a degree of preference. If we look at the Figure we see that AS2 may advertise a route for a prefix that originates in AS3 with some value of the MED field to indicate that the upper link is preferred. AS1, in turn, will use this information to determine the exit point for traffic to that certain prefix.

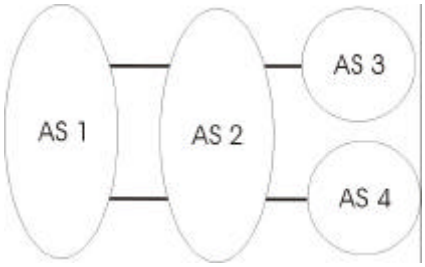


Figure 3-12. MED example [ST98]

LOCAL_PREF

Because an AS can have many entry and exit points, it is likely that it will receive multiple routes for the same destination. The value of the LOCAL_PREF path attribute can be used to indicate preferred route. This value can be assigned on the basis of some knowledge of a particular AS or some other information concerning the desirability of a certain (collection of) route(s). The LOCAL_PREF attribute is well-known and discretionary and shall be passed to all internal peers, but not to any external peers.

ATOMIC_AGGREGATE

The ATOMIC-AGGREGATE attribute allows BGP speakers to inform each other about decisions they have made with respect to overlapping routes [ST98]. A router might hear a set of overlapping routes from on of its peers, for example 138.39.0.0/16 and 138.39.12.0/24 of which the attributes are different so that they can’t be aggregated. If now, for some reason, the BGP speaker chooses to announce the less specific route, 138.39.0.0/16, it should attach the ATOMIC-AGGREGATE attribute to the prefix when it advertises it to other neighbours. When a BGP speaker receives an UPDATE with the ATOMIC-AGGREGATE attribute set, it must not take the prefix and de-aggregate it into any more specific entries. Also the router knows that traffic sent to an IP address associated with that route, might actually traverse other ASs than listed in the AS_PATH, because the more specific route might have a different value for this latter route.

AGGREGATOR
This is an optional transitive attribute which may be included in updates which are formed by aggregation. The AGGREGATOR field shall contain the AS number and IP address of the BGP speaker that performed the aggregation.

Policy routing and peering

As has been mentioned several times throughout this chapter, BGP allows for policy routing. In this last chapter the most important aspects of this will be summarised and peering will also be briefly addressed to, since both subjects have to do with organisational influence on the way the Internet traffic travels. Policy routing and peering influences the flow of information through the network. By establishing new links (peering) and by restricting or obligating certain routes, traffic will travel to its destination through other paths. The exact influence of routing policies and peering on routing and performance is very interesting for the UMEEPI project, but due to time considerations this falls outside the scope of this thesis research.

Policy routing

Policy routing has been previously defined as the ability to influence the routing information that will pass the border of an autonomous system. This in turn determines which traffic may pass this border. Note that there are other levels at which restriction to the traffic may be enforced, for example in firewalls, which are able to reject traffic from certain IP prefix ranges. Policy routing, however, operates at the level of autonomous systems. The policies are determined by the AS administration.

The main reason why BGP allows for policy routing is because of the AS_PATH attribute, which is not only used to avoid looping routing announcements. If a packet follows the route announced in a particular UPDATE message, with a particular AS_PATH, this packet will traverse the autonomous systems, listed in the AS_PATH attribute. This enables a router to perform policy routing. A router may decide to avoid a particular route, in order to avoid transiting a particular AS, which is useful for confidential information, for example. Another possibility is that a router has some information about performance of a particular (part of an) AS, which leads him to avoid this route. A third reason to avoid a certain route based on the AS_PATH could be that a route for the same prefix, with a shorter AS_PATH is available to the router. Typical routing policies involve political, security or economic considerations. Although BGP allows for policy routing, it is not part of the protocol itself, policies are manually configured into each BGP router[TAN96].

But also when traffic from outside the AS wishes to traverse the AS (transit traffic), resources are being consumed. Therefore, some transit ASs might place restrictions on the traffic that can traverse the network [RP00], and other ASs might close their networks entirely for transit traffic, these are called multihomed ASs. Huitema speaks of AUP in this context, meaning Acceptable Use Policy [Huitema]. For example the NSF Net AUP stated that only traffic related to scientific research or from research institutes could be allowed. The concept of AUP, leads to a classification of remote ASs in three categories: first class ASs can use the AS at will, second-class ASs can use the AS only to reach first-class ASs and third class ASs can't use the AS for transit traffic [HUI95]

The way in which these routing policies are enforced is by affecting the selection of paths from multiple alternatives and by controlling the redistribution of routing information and possibly altering the attributes of certain routes. This is symbolised in Figure 3-10 as the red circle. Some examples given in [RFC1772] are worth mentioning here:

1. A multihomed AS can refuse to act as transit AS by only advertising routes to destinations internal to the AS.
2. A multihomed AS can become a transit AS for a subset of adjacent ASs, by only advertising its routing information to this subset of ASs.
3. An AS can favour or disfavour the use of certain ASs for carrying transit traffic from itself.

Also some other performance related criteria can be controlled with BGP [RFC1772], such as routes with shorter AS_PATHs can be preferred over routes with longer AS_PATHs. Also, if by some means

information about the quality of transit ASs is available, this can be used to determine the most preferred routes, again by looking at the AS_PATH.
One final point of interest here is that BGP follows the "hop-by-hop" routing paradigm that is generally used throughout the Internet. This means that at every hop the best possible path is determined, which limits the control that a certain AS has over the paths taken by its traffic.

This also results in the fact that a BGP speaker only advertises to neighbouring autonomous systems those routes that it itself is actively using. In other words, if BGP speaker A advertises to its neighbour B that it has a path for reaching a particular IP prefix, B can be certain that A is actively using that path to reach the destination [ST98]. This has implications for the kinds of policy that BGP can support in the sense that any policy that conforms to this paradigm can be supported.

A problem with policy routing in this matter is that every AS can make its own decisions and that here can be policies that never converge [RP00]. It is a task of network managers to detect and correct these and other anomalies which makes BGP a configuration-intensive protocol [HUI95, RP00].

Peering

When two ASs decide that they will exchange traffic without using a transit AS, they make a **peering** agreement. In this case they have to have a direct physical connection as described earlier in this chapter, which can be highly expensive. With more and more ASs deciding to make peering arrangements, the physical infrastructure of the network changes. Once two ASs have a peering relationship they can agree to only use this link for traffic originating or destining in the neighbours, but they also might agree to provide transit service to one another. As we can see, this makes the organisation of the Internet even more complex.

The Amsterdam Internet Exchange (AMS-IX) is a nice example of a location where many peering relationships are possible. The AMS-IX has a very large ring-network connected by three switches, which is schematically represented in the Figure 3-13 [AMSIX].

When an AS is connected to this network it can peer with any other AS that is connected, without having to make an physical connections, because this is already there in the form of this ring network. For example, if AS 1 is peering with AS 2 their border routers are linked using the already available infrastructure. Because there are many ASs located in the buildings of the AMS-IX, they can also do **private peering**, fairly easy. In this case they do make a new physical connection. In the next Figure AS1 peers with AS2 and AS3, located in another building. Further more AS1 and AS2 have a private peering relationship.

Note that the switches provide for the Data Link Layer Connection, needed for an inter AS link, as described earlier in this chapter. Another point of interest is that the AMS-IX has AS number 1200, under which a specific IP prefix range resides. Because the BGP routers are directly connected to this infrastructure, they too reside under the AS number of the AMS-IX. However, when looked up via a WHOIS query, explicit remarks are made, showing the real AS number under which a specific BGP router, identified by its IP address, resides. So the border router of AS3, for example, has a IP address within the IP range of the AMS-IX, which suggests it being part of AS1200, but in reality it resides under AS3. The orange, dotted AS border shows this interesting situation.

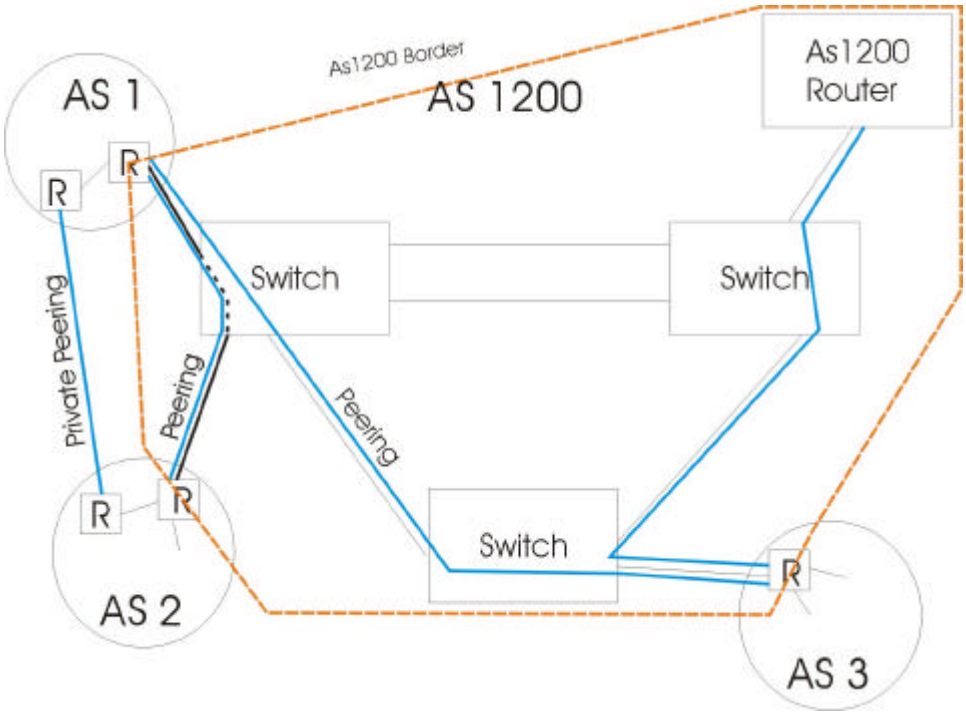


Figure 3-13. Amsterdam Internet Exchange example

4. Analysis and visualisation at the IP level

This chapter discusses the initial analysis of the RIPETT data and the visualisation of this data at the IP level. During these initial analyses and visualisation steps many ideas for further research where developed. Also practical uses of the data and the tools used where explored. Although these side-steps are very interesting and useful, only a few examples are displayed here. However, the practical ideas and recommendations for future research provided in this chapter are an essential result of this study.

First the RIPETT measurement principles will be briefly explained, along with an explanation of the traceroute tool. Then the data used in this part of the study will be presented, along with its limitations. Also some additional considerations to be made when working with this data are mentioned in this first paragraph. Next the initial analysis of the data will be presented. During this analysis it became evident that some custom program was needed to gain more insight in the data. This also provided the necessary practice with the data and the programming, because for the visualisation of the tracevectors also a program had to be developed. Note that only a small portion of all possible analysis and research could be performed within this research, but still this gives some useful insight in the chances and limitations of this kind of research.

Then the focus will shift from manual inspection and analysis to the visualisation of the data. The tools used and developed for this visualisation will be described and also some interesting results will be shown. More importantly, the use of these visualisations and the corresponding analysis will be shown, which will be done from two perspectives, one being an individual ISP and the other being a research group such as the group participating in the UMEEPI project. Individual ISPs are more interested in visualisations that might aid them in solving specific problems with their network and research groups are more interested in visualisations that might help them in gaining better understanding of the functioning of the Internet as a whole and in modelling the performance of the Internet.

RIPETT measurement principles

Because RIPE-NCC wants to deliver an objective measurement of the performance of the Internet, they have decided to focus on active measurements only. A test-box is connected as closely as possible to (one of) the border router(s) of each participating ISP (currently about 40). By connecting the test-box 0 hops away from the border router, most effects caused by the traffic inside the networks of the ISP are eliminated from the measurements. Because the test-boxes are connected to a GPS clock, the one-way-delay of a test message travelling from ISP-A to ISP-B can be measured accurately by sending a time-stamped packet.

The packets contain, among other information, the source, the destination and a sequence number, and are padded with random data to avoid compression by the networks that transfer them. Just before going to the socket the packet is time-stamped by the application layer. The receiver program collects the incoming packets and time-stamps them as soon as they are released by the Ethernet interface (in the data-link layer), thus eliminating further delays inside the receiving machine. In case a packet is fragmented, the receiver will use the time stamp of the last part as the arrival time of the packet.

By counting the number of packets sent and the number of packets that arrive, the packet-loss-rate can be determined. Parallel to the delay measurements, the test-box can determine the route to the other box using the tool **traceroute**, which will be described more thoroughly later in Chapter 4. These measurements take place from every active testbox to every other active testbox and thus in principle all paths between them at a specific moment in time are determined.

The measurements briefly described here will provide several observables, which can be stored in different matrices, which will be constructed on a central machine. One of these matrices contains the routing vector of path information between the two locations. This matrix provides, assuming that the test-boxes are installed at a significant fraction of the ISP's an up-to-date map of the Internet, as well

as the history of the network [UKKW98]. Another use of this data would be to verify whether a noticed change in delays between 2 test-boxes, could be explained by a noticed change in the routing vector between those 2 boxes.

With the observables described here, once enough data has been collected, meaningful metrics can be defined, such as 'one-way delay' and 'one-way loss'. The definition of the metrics is one of the topics being discussed in the IETF IPPM working group [IPPM].

Some final remarks must be made about the number of packets sent. This number must be high enough to detect fluctuations in the performance of the network that are interesting. But the amount of data generated by the test-traffic should be small compared to the bandwidth of the connection that is being tested, to prevent the measurement data affecting the performance of the network. Furthermore the measurements should be randomly distributed in time to prevent synchronisation of the test-traffic with other events on the network.

The Traceroute Tool

The traceroute tools is used to discover the route that an IP packet takes from a source to a destination, in our context called the routing vector.

The tool is based on the TTL field in the IP header. As each router handles a packet, it decrements the TTL field by one. When the TTL reaches zero, a router must discard the packet and send an ICMP error message back to the sender of the packet, the **TTL exceeded** message. This message contains the IP address of the router that sends the error message.

By initially sending a packet with the TTL set to one, the next hop along the path to the destination will receive the packet, set the TTL to zero and then discard the packet and return an ICMP message to the sender. Next the sender uses a packet with the TTL set to two. Now the second hop will set the TTL to zero and return the ICMP packet. By incrementing the TTL field in subsequent messages to a certain destination by one, each intermediate router will be forced to send its IP address back to the sender. The traceroute program will record these IP addresses and this way the path can be reconstructed.

A traceroute packet, like any other IP packet, also contains a port number, which indicates at which port at the destination the packet must be delivered. This port number is chosen in a way that it is highly unlikely that any host will ever accept any packets on the corresponding port. Therefor when a traceroute packet arrives at the destination it will send a **port unreachable** message back to the sender of the packet. For the sender (our traceroute program) this implicates that the packet has reached its destination and the process of increasing the TTL and sending another packet is stopped.

The traceroute tool can only receive the two described ICMP messages and in any other case it will treat the response in the same way, namely by placing an asterisks in the results, indicating that no hosts IP address is found.

There are several possible reasons for this to occur. Some routers, for example may place certain restrictions on ICMP messages, such as never to send any ICMP error messages, or only send a certain amount of ICMP messages in a given time interval. In this case no response will be received, the traceroute tool will try again for two more times and then it will increment its TTL and continue the path determination. This means that this entry in the trace is not resolved and an asterisks will be printed on the screen. However, also when a router doesn’t find the destination (“destination unreachable” message) or when a traceroute packet gets lost somewhere, the correct response will not be received, which will lead to a corruption of the trace in a similar way.

The RIPETT data

As we know, the data used for this study is comes from the RIPETT project, which has been briefly described above and in Chapter 1. This data consists of delays between two testboxes, which are determined approximately every 20 seconds and a list of intermediate hops between the two testboxes, which are determined approximately every 6 minutes by the traceroute tool. The lists of intermediate hops, also referred to as tracevectors, are important for this thesis study, since we try to gain insight in the topology of the Internet. The delays are more important to other studies performed within the UMEEPI project. Therefore the description and explanation of the RIPETT data will be limited to the tracevectors.

For the purpose of this study several dumps from the MySQL database of the entire dataset were made. These dumps were the starting point of this analysis and I will describe them here. This description will also give insight in the way the data can be interpreted and in some interesting considerations that were made while studying the data.

Each dump gives a snapshot of the valid tracevectors in a certain period of time. What is meant with the term valid will be explained later. Each dump is contained in an ASCII file with one line for each tracevector in the file. The following line shows an example of this:

```
tt01.ripe.net tt13.ripe.net 975622064 975625255 193.0.0.14
193.0.0.50 193.0.0.244 193.148.15.34 145.41.7.141 192.87.106.111
```

In this example tt01.ripe.net is the source testbox and tt13.ripe.net is the destination testbox. The IP addresses 193.0.0.14 through 145.41.7.141 are the intermediate hops and the last IP address is the IP address of testbox tt13.ripe.net. The two large numbers after tt13.ripe.net are Unix timestamps, which express the number of seconds since January 1st, 1970. These Unix timestamps give the starting and ending time of a certain interval in which this traceroute was valid.

In order to understand the term “valid” in this context it is important to know how the data in the database is collected. One of the considerations taken into account when setting up the RIPETT project was the frequency of the test traffic [UKKW98]. This frequency must be high enough to detect interesting fluctuations in the performance or to detect interesting changes in the chosen paths, but low enough to prevent that the test traffic would influence the performance of the network and make the measurements biased. Therefore the number of traceroutes that “happen” simultaneously must be small.

This is where another consideration is important. It was shown in [FJ94] that initially unsynchronised periodic routing messages could synchronise with each other and other events on the network, which means that the measurements could become corrupted. To prevent synchronisation to occur, a random component needs to be introduced in the sending of periodic messages. Within the RIPETT project this is done by using a Poisson distribution to determine the interval between two consecutive measurements from a certain testbox. For the tracevectors there are on average 10 measurements each hour [UKKW98], which means that approximately every 6 minutes a tracevector is determined between two endpoints.

Now the term valid can be explained. Whenever a traceroute command is executed, the starting time is recorded. It was observed that often the tracevector of two or more consecutive measurements was identical. In order to reduce the amount of data, it was chosen that time intervals would be used. In our example the interval is from 975622064 (Thu Nov 30 23:54:25 2000) to 975625255 (Fri Dec 1 00:10:44 2000), which means that between these two times each traceroute performed between tt01 and tt13 found the same tracevector. The number of traces performed between those two times is also recorded in the database and each different routing vector is given a unique ID. The fact that this interval ends on Fri Dec 1 00:10:44 2000 also means that the next measurement resulted in a different tracevector, because otherwise the interval would have been longer. The ending time of this interval is 00:10:44, which means that the starting time of the next interval is 00:10:45. In the collection of the data in the database it is assumed that a change in the tracevector between two endpoints occurs immediately after a traceroute. However, the exact time could be up to approximately 6 minutes later,

or it could have changed multiple times between two consecutive measurements, see the next paragraph for more on the limitations of the RIPETT data.

The data used by this thesis study consists of 6 datasets with the tracevectors that where valid at midnight on the first day of the last six months of 2000. So each dataset has one tracevector for each pair of the testboxes that were active at that time. Therefore each dataset gives us a snapshot of the network at the first day of the month, around midnight, given certain limitations explained in the next part of this paragraph. See appendix A for a small sample of the data from the dataset of 01-07-2000.

Limitations of the data

The approach of performing a traceroute every 6 minutes using a Poisson distribution solves the problems of synchronisation and overload, but it introduces several other consequences. Since the determination of intervals between two measurements is independent at each testbox we will never have a complete picture of the entire network at a specific moment in time. This can be easily understood by realising that it is highly unlikely that all tracevectors are determined at the exact same time.

Another similar problem that remains unsolved is that we will not know whether the route between two endpoints has fluctuated between two measurements. This means that we will not always know if a route is really static when we determine the same tracevector in several consecutive measurements. This is an important limitation if we want to use the data to study the dynamic behaviour of routing and routing tables. An interval of 6 minutes was chosen, rather arbitrarily, to prevent the traceroute measurements from influencing the performance of the network to much while still obtaining a reasonable view on the dynamics of the tracevectors. Also when we do find a change in the routing vector, we only know that the change occurred somewhere between two measurements, but we do not know if this was the only change that occurred within those approximately 6 minutes. Note that [ZHPA00] elaborates further on this subject, which is referred to as “routing stationarity”.

So although we can prevent some unwanted effects of our measurements (influencing the performance and synchronisation), we also loose some detail in the data using this approach. This, in turn, has its limitations on the kind of results we can obtain from studying this data. One thing we would like to know from a route between two endpoints, for example, is whether or not there are “oscillations” in the chosen path. This is also called “route-flapping”, which is the changing of the state of the link at short intervals. Each of these oscillations results in massive routing updates in each of the neighbouring ASs, which means major overhead and thus results in very poor networking service [HUI95]. The approach chosen in the RIPETT can only “see” route-flapping on timescales of more than 6 minutes. It may well be possible that this is not detailed enough to explain certain degradations in performance. If, for example, the delay between two endpoints increases in the interval between two tracevector measurements (note that delays are measured every 20 seconds), but the tracevectors found in those two measurements are the same, it would be tempting to state that the delays are not caused by route flapping. However it might well be possible that a route changes 10 times before arriving back at the previously measured route before the second tracevector measurement is performed.

Some other limitations of the data are:

- Not all testboxes are active at every time, either on purpose or by accident. This makes it difficult to compare results from analysis of the different datasets. This also causes the great differences in the number of tracevectors in the datasets. For example in the 01-10-2000 dataset only 27 testboxes where active around midnight, which resulted in 702 tracevectors (n*n-1).
- The limitations of the traceroute tool, such as routers configured not to give ICMP messages, also have their consequences in the data. If the traceroute program does not receive the correct ICMP messages, as explained in the previous chapter, an asterisks is placed on the screen. In the RIPETT implementation of the traceroute, instead of an asterisks, the 255.255.255.255 IP address is used. Of course this corrupts the data in a certain way, which makes it less straightforward to work with the data. Some of these and other errors can be corrected, but this is not the focus of this study. Rob Meijer from KPN research is developing algorithms to tidy up the data and restore errors like these, but the subject seems difficult to grasp.

- Another limitation of the traceroute tool is the maximum TTL of 30, which means that if a path is longer than 30 hops the rest of the path can not be found. In the RIPETT data this results in the last hop in such a tracevector not being a testbox, but the 30th hop in the trace.
- The data used is only a small portion of the available data in the RIPETT project. This makes it difficult to study dynamic behaviour at different time scales. For this thesis study only six datasets with one month intervals are available. This, however, has still provided us with many useful insights and ideas.

Special or reserved IP addresses

Before analysing the RIPETT data any further it is important to have some more knowledge on the IP address space and on special or reserved IP addresses. If certain addresses are found in the data, this suggests some configuration error in the router or corruption of the data in some other way. Although it is not our intention to correct these errors, knowledge about this is still useful. This way some general characterisations of the correctness of the data can be made.

We've already seen in Chapter 3 that certain addresses containing all zeros or all ones are special addresses. Hosts use the 0.0.0.0 address when they are being booted and it refers to "this host". When the network prefix is non-zero, but the host number is all zero's "this network" is meant. The all zeros network prefix is used to refer to "**a** host on this network", for example 0.0.a.b means host a.b on this network. Next the all one's address 255.255.255.255 is reserved for a broadcast on the local network. When the network prefix is not all ones, but the host number is, a "broadcast on a distant network" is meant. For example if a.b.255.255 is used, this means a broadcast on the network with the network prefix a.b. Finally addresses of the form 127.x.y.z are mainly used for "loopback" testing. Recall that when the traceroute program used within the RIPETT project doesn't receive a valid IP address from a certain host, also the 255.255.255.255 address is used, indicating "host unknown".

Along with these special addresses there are many reserved addresses, which are not assigned by the IANA and the Internet Registries. So also, if these addresses are encountered in the data, something is "wrong". An example of these addresses are the three blocks that are reserved for private internets: 10.0.0.0 - 10.255.255.255 (10/8 prefix) 172.16.0.0 - 172.31.255.255 (172.16/12 prefix) 192.168.0.0 - 192.168.255.255 (192.168/16 prefix) [RFC1918]. Note that (in pre-CIDR notation) the first block is nothing but a single class A network number, while the second block is a set of 16 contiguous class B network numbers, and third block is a set of 256 contiguous class C network numbers.

Analysis

Many statistical and mathematical analysis has been made on traceroute data [ZHPA00, MHH00], and the focus of this study will not be on this part of internet measurements and modelling. This study focuses primarily on the practical use of this data for network operators and test traffic measurement projects. The approach chosen was visualisation, but still some initial exploration of the data had to be performed. This analysis focuses on some common errors in the traceroute data and on some simple characteristics of the datasets used. This has lead to some more insight in dealing with data like this and also to many ideas and recommendations for the RIPETT project or similar work.

Hop Count

One of the more straightforward statistics of a set of traceroute is the average hop count, which describes the average “distance” between any two endpoints in the Internet, where distance means the number of hops needed to travel from the source to the destination. But not only the average, but also the histogram of the hop count is interesting, because this can be used to discover certain anomalies in the data. This initial analysis was performed by simply using a spreadsheet program. The values that were found during this initial exploration of the data could later be used to check the developed program. Figure 4-1 show the histograms that were made from the six datasets.
From the analysis and these histogram we see that the average hop count is somewhere between 14

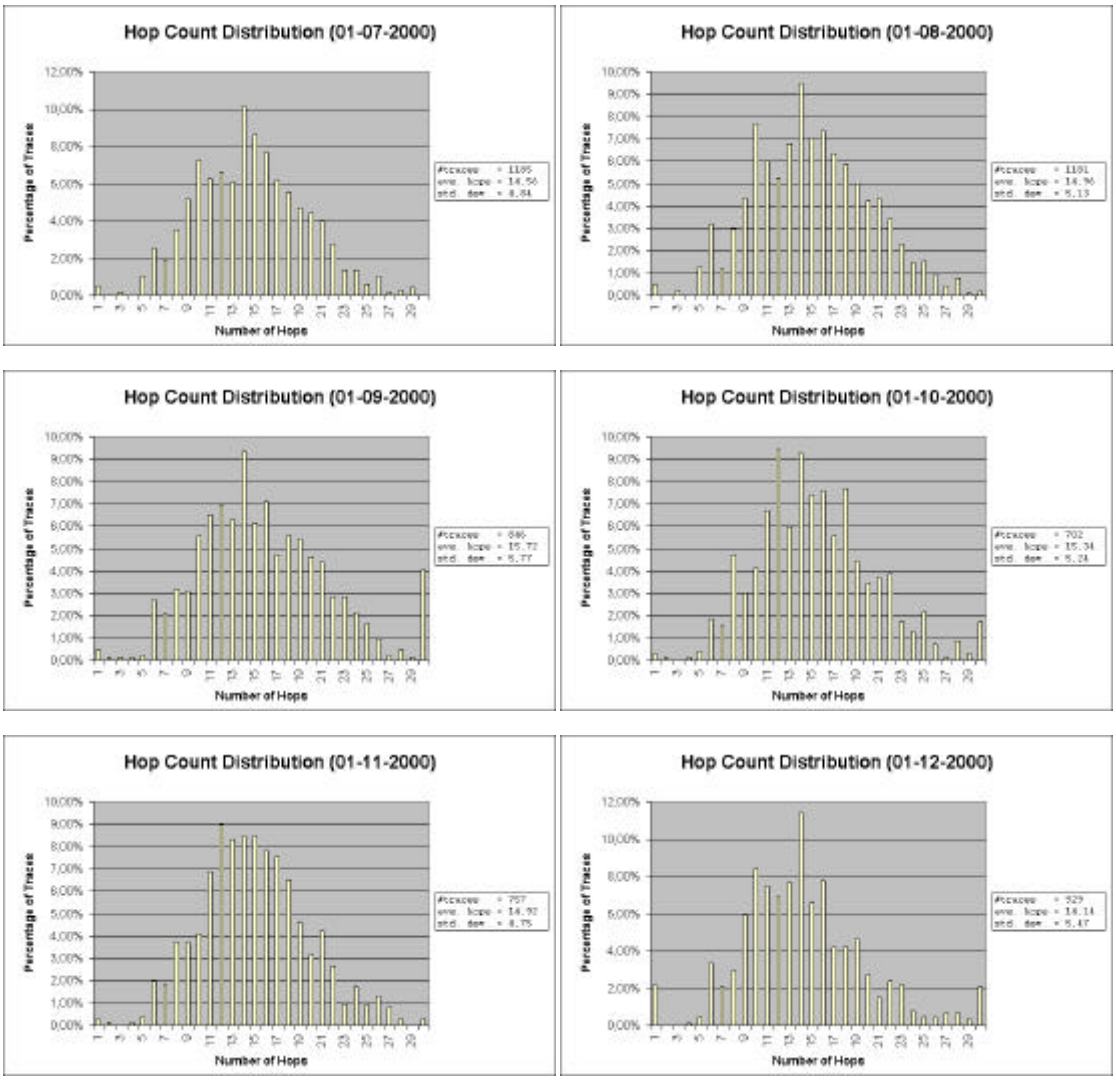


Figure 4-1. Histograms of the six datasets

and 16 for all 6 datasets. These values are in agreement with values found in other measurements [MHH00, PA97]. Further more we see that the highest peak is placed somewhere near this average, indicating that indeed most traces (10% for the first dataset) have this average number of hops. Also when combining all the traces in the datasets in one dataset, analysis finds the average of 14,9 hops, which is a value that agrees with the results of other researchers within the UMEEPI project. The histogram of this can be seen in Figure 4-2.

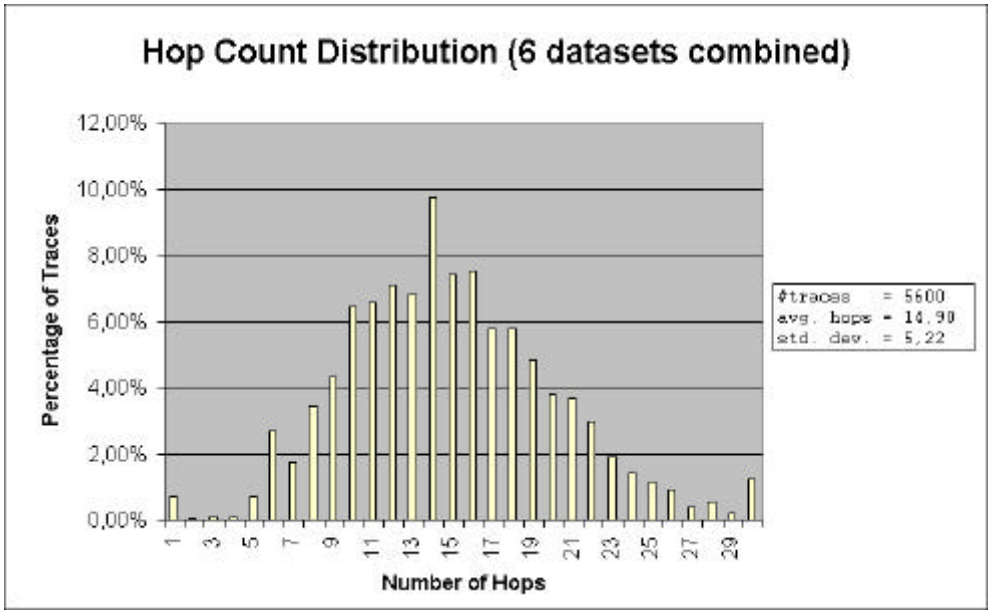


Figure 4-2. Histogram of the combined data

The results shown in Figure 4-2 raised a suspicion concerning the 30 hop limit used with the traceroute tool, because there is a strange peak at 30 hops. This called for some additional analysis which is described next in this chapter. This description also gives an idea of the complexity of the subject and also some interesting discoveries that can be made when looking closely at the tracevectors.

The 30 hop limit

Within the RIPETT project, the traceroute tool is configured to allow for a maximum of 30 hops in each trace, which has already been mentioned in the limitations of the data paragraph. However, this is only a real problem when this happens a lot and thus corrupts the data too much. Also when a certain combination of testboxes (source and destination) systematically leads to a tracevector of 30 hops, where the last hop is not the address of the destination testbox, this poses a problem. Here we try to gain some more insight in this matter.

The strange peak at the 30 hop point in Figure 4-2 could be caused by the fact that all traces of 30 hops and over (30+ hop traces) are recorded as 30 hop traces, thereby placing to much weight on the 30 hop point. If this is the case, then the method of data collection needs to be adjusted, because there are too many 30+ hop traces. This also has consequences on the average hop count. However, closer analysis shows that there is probably a different cause for this high peak.

We can see that in the three histograms of the datasets from 01-09-2000, 01-10-2000 and 01-12-2000 there are exceptionally high peaks at the 30 hops point, with values of 4%, 1,7% and 2% respectively. Of course this influences the average hop count in some way. Since only these three datasets show these extreme peaks at the 30 hops points, it is likely that some special situation that occurred at those moments is the cause of the high peak. But it might also be possible that traces to certain

testboxes are often longer than 30 hops and that these testboxes where not active at the time of the other three datasets. This calls for a closer look at the 30 hop traces in the three datasets.

Let's take the most recent dataset (01-12-2000) as an example. In this dataset we find nineteen traces with 30 hops, which is 2% of all traces. Table 4-1 shows the source and destination testbox of these 19 traces. The source – destination pairs are ordered in such a way that the left part of this table contains traces were the 30 hop limit is exceeded in both directions. The traces in the right part of this table don't show this symmetry and the 'return paths' of all these trace have less than 30 hops. For these 9 traces, the number of hops in the return path is also given.

Source	Destination	Source	Destination	'return path'
Tt17.ripe.net	tt19.ripe.net	tt36.ripe.net	tt47.ripe.net	23 hops
Tt19.ripe.net	tt17.ripe.net	tt44.ripe.net	tt08.ripe.net	16 hops
Tt44.ripe.net	tt19.ripe.net	tt44.ripe.net	tt12.ripe.net	25 hops
Tt19.ripe.net	tt44.ripe.net	tt44.ripe.net	tt20.ripe.net	19 hops
Tt46.ripe.net	tt19.ripe.net	tt47.ripe.net	tt07.ripe.net	26 hops
Tt19.ripe.net	tt46.ripe.net	tt47.ripe.net	tt08.ripe.net	20 hops
Tt28.ripe.net	tt47.ripe.net	tt47.ripe.net	tt12.ripe.net	28 hops
Tt47.ripe.net	tt28.ripe.net	tt47.ripe.net	tt20.ripe.net	26 hops
Tt22.ripe.net	tt44.ripe.net	tt47.ripe.net	tt22.ripe.net	26 hops
Tt44.ripe.net	tt22.ripe.net			

Table 4-1. The 19 tracevectors from the 01-12-2000 dataset that contain 30 hops.

Upon closer examination of these tracevectors we find we find that the first eight in the left part of the table end with a long series of 255.255.255.255 addresses. This suggests that somewhere along the route the traceroute program did not receive the correct ICMP messages and that this went on, until the 30 hop limit was reached. The first six of these traces involve testbox tt19.ripe.net, three times as source and three times as destination. Upon closer examination of these traces we find that when tt19 is the source, the series of 255.255.255.255 addresses starts very early in the tracevector, and when tt19 is the destination, they start at the end of the tracevector. This makes it likely that there is some error close to testbox 19 in a router or the internal routing of a subnet. The next two tracevectors (without timestamps) are taken from the dataset and illustrate the situation close to testbox 19:

```
tt19.ripe.net tt17.ripe.net 192.115.187.97 212.116.160.50
255.255.255.255 255.255.255.255 255.255.255.255 255.255.255.255
255.255.255.255 255.255.255.255  (…)

t44.ripe.net tt19.ripe.net 205.168.101.254 208.46.65.65
208.46.65.65 205.171.5.35 205.171.18.9 205.171.5.121 205.171.22.9
 205.171.5.153 205.171.19.38 205.171.4.150 207.45.220.97
207.45.223.61 207.45.221.102 255.255.255.255 207.45.216.138
192.115.106.246 212.179.73.238 212.116.160.49 255.255.255.255
255.255.255.255 255.255.255.255 (…)
```

In all six of these traces close to tt19 we see that the last known IP address starts with 212.116.160, indicating that there is something wrong with the internal routing of a subnet, or maybe only with one router that has multiple IP addresses within this IP range. Although the precise error can't be derived from this data, it is almost certain that there is something wrong close to testbox 19 that needs closer inspection. These kinds of analysis provide valuable information for troubleshooting the network of an ISP. With the traces between testbox 47 and testbox 28 a similar situation occurs, where the tracevector breaks of close to testbox 28.

The other eleven tracevectors in the table do not contain any 255.255.255.255 entries, so something else is going on. One of the first things that is notable is that in each of these tracevectors, either tt44 or tt47 are involved as one of the endpoints. Testbox 47 is located in New Zealand and testbox 44 is located in Denver, which lies in the middle of the United States and thus far from the coast and the cross-Atlantic links that connect the US to Europe. In all these tracevectors the other testboxes are somewhere in Europe, which makes it intuitively more likely that many hops are needed. Still, in order

to be certain of these kinds of suggestions, some theory has to be developed that relates physical distance to hop count, which is one of the goals of the UMEEPI project, but falls outside the scope of this study. However, the fact that all these >30 hop tracevectors are intercontinental gives us some evidence that geographical location and physical distance do have at least some influence on the number of hops.

Another thing that comes to attention is that the return paths are shorter, which shows the asymmetry in chosen paths between two endpoints. This partly undermines the previous explanation relating physical distance to hop count, and it is required to look for other explanations for the large number of hops. One possible explanation is the existence of routing loops or other errors. Although very interesting, this problem remains unsolved for now and this illustrates that when learning more about this kind of data and research, more and more questions arise.

To finalise the subject of the 30 hop limit, lets look at the other datasets. We see several interesting things. The peak in the histogram of the 01-09-2000 dataset is mainly caused by the fact that all traces to testbox 28 resulted in an endless list of 255.255.255.255 addresses. This indicates that at that time there was something wrong close to testbox 28, that corrupted these trace vectors. This is similar to the example of the tt19 testbox in the 01-12-2000 dataset, but here only the traces **to** tt28 resulted in such a tracevector. This accounts for 29 of the 34 suspicious traces. Two other trace vectors where also filled with 255.255.255.255 addresses, which leaves only three traces where the cause of exceeding the 30 hop limit is less obvious. With the 01-10-2000 dataset, this is a little different. Of the 12 30+ hop trace vectors, only for are caused by a long series of 255.255.255.255 entries. So for these traces, further analysis would be needed to determine the cause of this large number of 30+ hop tracevectors.

One final point of interest is to look at the 30 hop limit, when we leave out the tracevectors that end with a series of 255.255.255.255 entries. The total available data contains 68 of these traces and when filtering these out we see that only 23 tracevectors (0,42%) with 30 hops remain . Also the average number of hops drops from 14,9 (std. dev 5,2) to 14,8 (std. dev. 5,1).

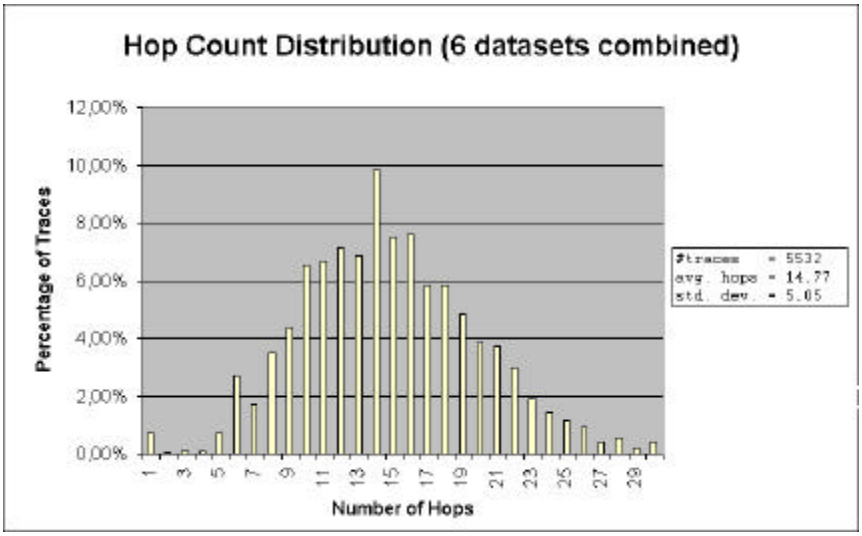


Figure 4-3 The total data, without the corrupted (255.255.255.255) tracevectors

When examining the remaining 23 tracevectors we find that only 2 of these tracevectors have the destination testbox as final hop, indicating that these tracevector is indeed exactly 30 hops. 10 Traces contain loops or unknown hosts, which might mean that the tracevector in reality might be actually less than 30 hops. This still leaves 11 tracevectors where the trace is ended after 30 hops, before reaching the destination. Six of these tracevectors involve testbox 47 (New-Zealand), and the other five involve testbox 46 (four times) and testbox 44 (once), both located in Denver, US.

Parsing the data

After this initial analysis using a spreadsheet it became evident that a custom program was needed to handle the data and to be able to give some more useful insights. The Java language was chosen for this programming effort because different platforms are used at the different involved parties. The result of the initial programming effort was a program that could read in the data from the RIPETT project and give some useful statistics of this data. For a summary of the results found by this program see Table 4-2.

The program parses the tracefiles, which are normal ASCII-files, by reading each line. While reading, the unique IP addresses are stored in a list and each IP address is counted. Since the syntax of each line is known, some simple calculations are possible. The total number of hops, for example, is the same as all the counted IP addresses, because the source is not represented in the list of IP addresses. Also by subtracting the number of traces from the number of hops, we get the total of intermediate IP addresses. This is because for every trace, the last IP address in line is the destination, so one IP address has to be subtracted.

Note that this introduces some errors for the cases where a trace is longer than 30 hops, because there the last IP address might not be a testbox. However, as we've seen in the initial analysis, this is only a small fraction of the traces and this is not corrected in this version of the program. Still, it would be possible to compensate for this by counting the number of traces with more than 30 hops, where the last IP address is not a testbox, and adding these to the total of intermediate IP addresses. But then it would be also necessary to correct other errors before counting, such as verify the last hop of every trace, handle 255.255.255.255 traces, detect routing loops and so on. The approach taken throughout this research is not focused on correcting errors and giving 100% accurate results, but rather on gaining as much insight as possible from the data we do have. The research of Rob Meijer, also part of the UMEEPI project, primarily focuses on correcting corrupted data [ME01].

Because the number of unique IP addresses was counted, it is possible to determine the number of duplicate entries. Finally, the maximum number of hops encountered in a trace is also recorded and the number of unique source testboxes and the number of unique destination testboxes are counted from the first two strings in each line of the trace file. The average number of hops is calculated by dividing the total number of hops by the total number of traces.

	01-07-00	01-08-00	01-09-00	01-10-00	01-11-00	01-12-00	combined
# traces	1185	1181	846	702	757	929	5600
# sources (testboxes)	35	35	29	27	28	31	39
# destinations (testboxes)	35	35	34	27	28	31	39
# intermediate IP addresses	16068	16481	12451	10066	10541	12205	77812
Duplicate	13802	14180	10635	8358	8744	10446	73553
Unique	2266	2301	1816	1708	1797	1759	4259
# hops	17253	17662	13297	10768	11298	13134	83412
Avg. hops	14.56	14.96	15.72	15.34	14.92	14.14	14.90
Max. hops	29	30	30	30	30	30	30

Table 4-2. Summary of the results of the program

When comparing the results of different datasets, we encounter some interesting things. First we see that the number of active testboxes varies greatly, which has its impact on the number of measured trace vectors in a certain period of time. Among the reasons for a testbox to become in active are that people mistakenly place filters on ICMP messages, the testbox is disconnected for transport, or just turned off. In one file (01-09-2000 dataset) the number of receiving testboxes differs from the number of sending testboxes. Here 29 of the 34 testboxes that where active could actually send traceroute packets and measure the tracevector to 34 destinations. Five testboxes could only respond to traceroute packets by sending ICMP messages, but no traceroute commands where performed from these testboxes, or at least there is no record of this. This too can be explained by people mistakenly placing filters or limits on ICMP messages, or maybe the testbox was placed behind a firewall.

Further we see that the average hop count varies between 14 and 16, which was already found in the initial analysis.

The most interesting value determined with this first program is the total number of unique IP addresses. As we've seen, each IP address corresponds to an interface on a host or router. So, in our effort to visualise the Internet, these unique IP addresses are essentially the nodes of our graph. What we see now is that at a specific moment in time, for example 01-07-00 around midnight, 2266 unique nodes are part of the graph that connects the 35 testboxes to one another. The fact that 1185 unique traces only contain 2266 unique nodes raises the thought that many traces overlap. Each IP address participates on average in 1,9 traces, but this doesn't really say much. It is much more likely that there are many IP addresses that only participate in one trace and that there are some IP addresses that participate in many traces. To study this the number of occurrences of each unique IP address in the dataset needs to be examined. This will also give insight in the importance of a certain node / IP address. Due to time limitations of this thesis study this has to remain a recommendation for further research.

Another use of this value (# unique IP addresses), is to gain some insight in the dynamics of the routing in the network, without having to visualise this. We see that each dataset contains about 2000 unique nodes. However, we can determine that these 2000 nodes are not always the same in each dataset. Moreover, if we collect al the data in one file and count the unique nodes in it, we find around 4000 unique IP addresses. This shows, that there is at least some dynamics in the routing between the testboxes, and that at least some part of the paths chosen at one time differ from those chosen at another time.

Now that some initial analysis has been made and some feeling with the data and insight in the many different possible analysis is gained, the focus will shift to visualisation of the tracevectors. Visualisation at the autonomous systems level is an important research goal, and visualisation at the IP level is a necessary first step, which also provides us with additional insights. The next paragraph shows how the data is visualised and what tools where used and developed, before showing some visualisations and their use for both the individual providers and the research project concerning internet measurement and modelling, such as the UMEEPI project.

Visualisation

Although many ideas for further analysis have been found, now the focus will shift to the visualisation part. A previous study [NICK] has focused on the visualisation as a graph and the associated layout algorithms itself, but here the focus lies on the practical use of visualisation. Still, the actual visualisation has to be performed also, and it was decided to use an existing visualisation tool called, Otter [HNC99]. This reduced the programming task to exporting the data to the import format of the Otter program. First the Otter program will be briefly described, and then resulting program of this phase of the thesis study, NodeLink 0.7 will be described. After this some visualisation and their use to a single ISP and to research projects such as the UMEEPI project are shown.

Otter 0.9

Otter is a tool created by the Cooperative Association for Internet Data Analysis (CAIDA) for visualising arbitrary network data that can be expressed as a set of nodes, links or paths. This of course also includes data on topology and routing [HNC99].

Otter uses a specific input file format which contains the data, but can also contain a data specification indicating specific attributes of each object in the data file, which Otter can use to tailor its visualisations and to adapt its user interface. This data independence is the main strength of the Otter tool. But also the visualisation algorithms are interesting. This data independence is achieved by using a specific input file, which accepts many different options and attributes, as long as it satisfies certain given specifications.

The layout of a graph involves two phases: first the subset of the root nodes and hen positioning of the other nodes, relative to these root nodes. The concept of root nodes comes from the desire of the programmers to render central positioning to a specific subset of the nodes.
The two most important layout options for the root nodes are straight topological, which places the root nodes on a circle in the centre of the screen, and (semi-) geographical, where the nodes with additional geographical information can be placed on an imaginary map. Some additional information about these layout algorithms is given in the description of Otter [HNC99].

When the root nodes are placed, the non-root nodes without geographical information are placed in semi-circles around their parent node, using a breadth-first scan from the roots and guided by the following heuristics [HNC99]:

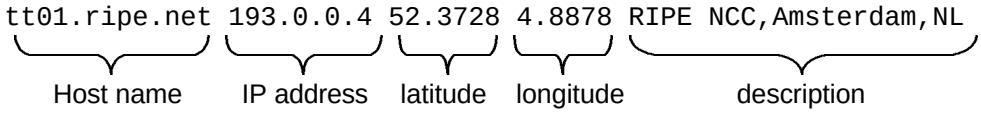
1. the more children a node has, the farther away those children should be so that they will not overlap
2. children that themselves have lots of children should be even farther away so that the grandchildren do not overlap

Of course the nodes can also be placed manually to make other or more useful representations. However these automatic layout algorithms are essential to be able to work with larger datasets and still maintain overview. Many other functionality has been put in to Otter, like the ability to use colour and attributes for nodes and links. One possible use of this is to colour the links in relation to the delay that occurs on those links, or colour specific paths within the graph of the entire dataset. These options are very interesting and useful, but for this thesis study focus will be on the visualisation of nodes and links without any further enhancements.

NodeLink 0.7

When the choice for using Otter was made, the programming task was to collect all the nodes and the links of the graph from all tracevectors, before exporting this to the Otter program. This resulted in the Java program, which was called NodeLink. Here a brief description of this program is given, and the source code can be found in Appendix B.

The first step of the program is to read in the list of testboxes from a text file. This is needed, because the program needs to be able to identify the testboxes by IP address, as well as by name. Also the geographical location of the testboxes can be exported to Otter, so these have to be known by the program. A user-editable text file was chosen, because some characteristics of a testbox might change in the future, e.g. the geographical location or the IP address. An example of a line in the testboxes.txt file is shown below. A complete list of testboxes is shown in Appendix F.



The testboxes are stored so that their characteristics can be retrieved both by using their hostname and their IP address.

The next step is to read the traces from an input file. These input files have the same syntax as the original data files provided by the RIPETT project and in fact these data files can actually be used. However, also modified files can be used, as long as they have the same syntax. This is useful for making a specific visualisation or analysis. This way it is possible to select only those traces that comply with certain conditions, for example, only those traces to a certain testbox at a specific moment in time.

The parser used is quite simple and the correct syntax of the input file is very important. The parser will also check whether all testboxes in the input file are known from the testboxes.txt file, and prompts the user to update this last file, in case an unknown testbox is encountered.

After reading in the trace vectors some statistics are collected. These are limited to the total number of traces, the total number of hops and the total number of unique hosts. The average number of hops per trace are calculated and the unique hosts are stored in a vector.

Next, the program asks the user if filters on the traces should be applied. Three filters are implemented:

1. A filter that checks a tracevector for any loops and discards the tracevector if a loop is found
2. A filter that checks whether an unknown hop (255.255.255.255) is present in the tracevector and discards the tracevector if it is found.
3. A filter that checks whether the last hop in the tracevector is a testbox and discards the tracevector if it is not.

The use of these filters complies with the approach in this study to refrain from correcting errors and anomalies in the data and to focus on the information we do have. So instead of trying to correct the data, suspicious traces are just left out. After the use of these filters, the statistics are again collected.

At this phase in the program's execution, we have the trace vectors and the unique hosts stored. The next phase is to build a graph and export it to the otter data format, stored in a *.odf file. The nodes are taken from the list of hosts and the links are determined by taking each trace and examining the subsequent hosts. The first time that a link between two hosts is confirmed it is stored. The next time this link is found it will be ignored because it is already there. Note that some support for bi-directional and uni-directional links is already written, but this is not yet fully implemented, so links are undirected at this time. Finally the graph is exported to the odf format. For details, please examine the source code in appendix B.

Visualisation of entire network

This paragraph shows some examples of typical visualisations that can be made using the developed tool. Here the focus will be on visualisations that are useful for our research and modelling efforts, thus the visualisations of the entire network at a specific moment in time. The resulting graphs and some statistics about them are not very interesting for a single ISP or network operator, but they do provide some insight in the problems faced when trying to understand and model the behaviour of the Internet as a whole. Here only a few of the visualisations are shown, for more visualisations see Appendix D.

The first visualisation that was made was that of the 01-07-2000 dataset, which contains 1186 traces. None of the filters where applied and the resulting graph shown in Figure 4-4 contains 2302 nodes and 3717 links. It is clear that this is a very complex graph. The graph layout algorithm is “straight topological”, which has been explained in the paragraph about Otter.

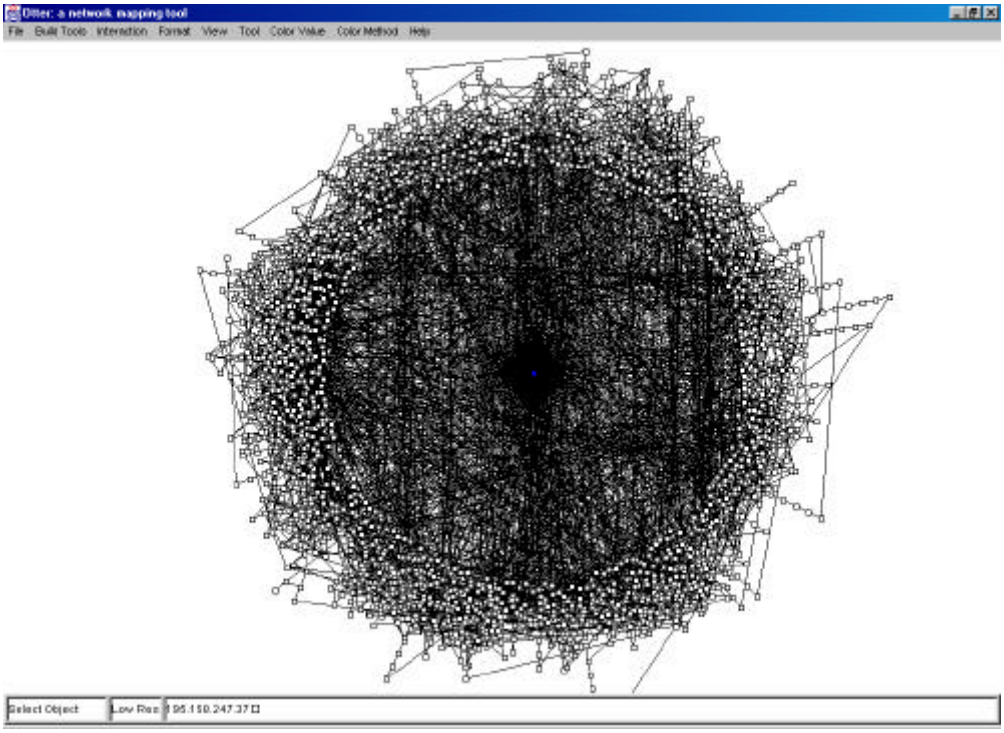


Figure 4-4. Graph of the 20000701 dataset

With this first visualisations also the first problems are encountered. One of these problems is that some tracevectors contain an 255.255.255.255 entry, as explained before. When visualising these tracevectors this 255.255.255.255 is treated as one single host and therefor as one single node in the graph. Since this address appears in many tracevectors adjacent to many different IP addresses, the 255.255.255.255 node in the resulting graph is highly connected to the other nodes. This corrupts our visualisation, because in fact there is no host with the 255.255.255.255 address and this node represents many different and unique other hosts. When placing the nodes following the semi-geographical layout, as is done in Figure 4-5 the influence of this 255.255.255.255 node becomes even more evident.

The testboxes are marked red in this second graph of the 01-07-2000 dataset and we see that a large portion of the nodes is placed around he 255.255.255.255 node, which illustrates the influence this node has on the graph. One interesting detail is that all connections to the New-Zealand testbox go through this cluster, which, however, doesn’t mean that all traces from and to this testbox contain a 255.255.255.255 address, as we can see in the Figure 4-7 where the unknown hop filter is applied.

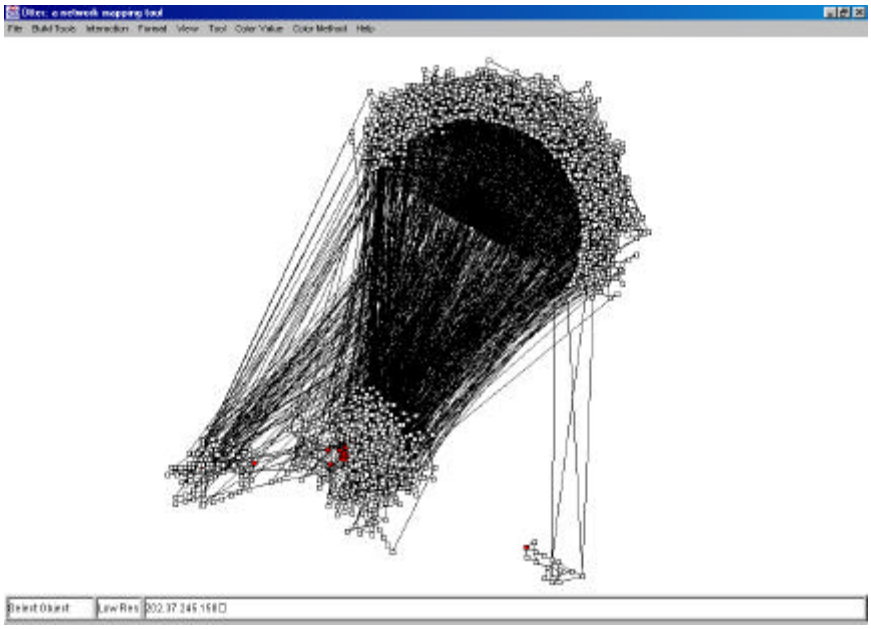


Figure 4-5. Graph of the 20000701 dataset in semi-geographical layout

It is clear that something has to be done about this situation and there are essentially two types of solutions. The first is to try to correct each and every tracevector that contains a 255.255.255.255 entry. This would require some algorithm that determines the actual IP address of this host, which is a complex subject that Rob Meijer is working on [ME01]. Another way to correct a trace vector would be to skip an unknown hop and simply make a link between the previous and the next hop as if they are actually connected, see Figure 4-6. I'm sure other creative ways to deal with this problem can be developed, but it would always introduce other uncertainties and more over it would distract too much from the actual problem of gaining insight in the topology and performance of the Internet. Therefore the second type of solution is chosen within this thesis study, namely to just discard of these "strange" tracevectors and focus on the information we **do** have.

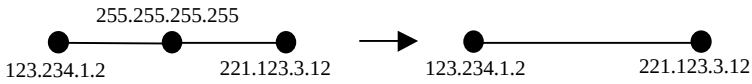


Figure 4-6. One of the ways to deal with unknown hops

When applying the filter on the first dataset we loose 195 traces, which is about 16%. However, when looking at the level of the graph, we loose 169 nodes which is only 7%. Now it might seem that this is only a small figure, but we do not know how many additional nodes we would have found if we where able to correct these entries.

To see this, consider that only one of these 169 nodes that are removed from the graph is actually the 255.255.255.255 node. All 168 other nodes correspond to hosts that are only encountered in the 195 tracevectors that where removed. The 20000701 dataset contains 577 entries of the 255.255.255.255 IP address, which might mean that in reality we miss up to a maximum of 577 unique nodes by leaving out the 255.255.255.255 nodes and another 168 for discarding the entire traces that contain them. Instead of 2302 nodes our graph would then contain 3047 nodes and we would loose 745 of these nodes, which would be about 25%. Although this latter value is probably over-estimated, we can only say that we loose somewhere between 7% and 25% of our nodes and their corresponding links. Depending on whether or not this would be highly connected nodes, this would significantly corrupt our view of the network. Similar calculations are valid for the other dataset and this shows a clear limitation of this initial visualisation effort.

Figure 4-7 shows the 20000701 dataset with the unknown hop filter applied, and we see clearly that the graph is a little less crowded. The geographical layout is shown below. When looking at this geographical layout it might seem that some nodes are placed in New Zealand, but apart from the testboxes, no implications about location of nodes are made, the nodes are just placed close to their parent nodes as explained in the beginning of this paragraph.

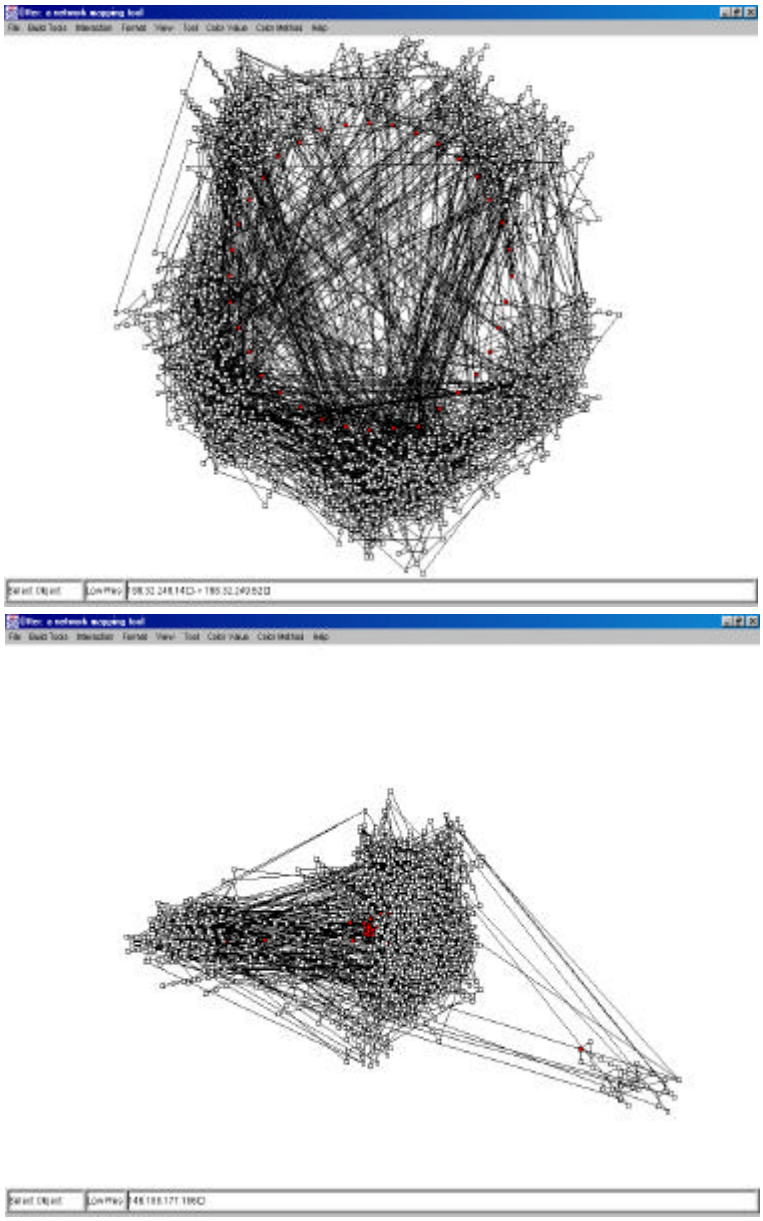


Figure 4-7. 20000701 dataset after unknown hop filter. Top: straight topological. Bottom: semi-geographical

Next to the 255.255.255.255 entries in the dataset, there are some more situations where a tracevector is unreliable. Filters for two additional kinds of tracevectors have been added to the program, one for tracevectors that contain loops and the other for tracevectors where the last hop is not a testbox.

If a tracevector contains a loop, then there is a routing error somewhere along the path, or an error occurred while recording the data. Either way, something is wrong with the trace and something has to be done with it. Ideally there would be some kind of alarm whenever a routing loop is recorded, so that network engineers can use this to troubleshoot the routing. Also studying these traces and trying to

determine where exactly and how often these loops occur would be very interesting and useful, since testbox hosts could use this kind of information to avoid routing traffic toward an ISP where many routing loops occur, for example.

Another strange situations occurs when the final hop of a tracevector doesn't seem to be a testbox. This can have several reasons, one of which is that it might be a 30+ hop tracevector that in fact was longer and caused the tracevector to be cut-off at the 30th hop. This could also happen when the IP address of a testbox has changed and when this is not updated in the testboxes.txt file. Also, when a trace stops for some reason before it reaches the destination this might occur. When a last hop is not a testbox the trace will not really add any value to the graph, since we are interested in connectivity between the testboxes. Although it might still give some additional information about connectivity of intermediate hops.

One problem encountered with the use of these filters was that the order in which they where applied gave different results. This can be understood by realising that when a trace ends at a series of 255.255.255.255 entries, this trace falls into all three categories mentioned above. It has a loop because the 255.255.255.255 occurs multiple times, and the last hop (255.255.255.255) is not a testbox. For the resulting dataset this doesn't matter, but if we would like to count these special situations this presents us with some problems, since we can't really differentiate between certain situations. In the current implementation the "last hop is no testbox" filter is applied first, then the loop filter and finally the unknown hop filter. However, applying the different filters separately and combined, some insight in the real values can be gained. This has lead to the following statistics for the 20000701 dataset.

	No filters	255… filter	Loop + 255 filter	Last hop no testbox filter	All filters
# loops filter	0	0	167	0	167
# unknown hops filter	0	195	88	0	88
# "no testbox" filter	0	0	0	0	0
# remaining traces	1185	990	930	1185	930
# nodes	2302	2133	2085	2302	2085
# links	3717	3222	3104	3717	3104

We see in the table that 167 traces contain loops, but when this filter is applied, we see that only 88 traces with unknown hops remain. This indicates that 195 – 88 = 107 of the 167 traces with loops also have 255.255.255.255 addresses in them. Therefor it is not clear whether these 107 tracevector actually have loops in them or whether this is caused by multiple 255.255.255.255 addresses in the tracevector. However, it is clear that at least 167 – 107 = 60 traces actually do have loops. These 60 traces would be interesting to study further, which will not be done here, due to time restrictions. As a second example let's look at the 20000901 dataset in the following table.

	No filters	255… filter	Last hop and 255 filter	Loop + 255 filter	Last hop no testbox filter	All filters
# loops filter	0	0	0	90	0	56
# unknown hops filter	0	135	103	57	0	57
# "no testbox" filter	0	0	35	0	35	35
# remaining traces	846	711	708	699	811	698
# nodes	1850	1669	1663	1651	1798	1645
# links	2748	2408	2398	2380	2655	2371

When only the unknown hop filter is applied we find that 135 traces are removed from the traces. When also the last hop filter is applied, we find 35 traces where the last hop is not a testbox and 103 traces with unknown hops. This means that 135 – 103 = 32 of the 35 traces where the last hop is not a testbox are traces that have the 255.255.255.255 address as last hop. From the initial analysis we know that 31 of these are also 30+ hop traces, so one of these traces that ends with 255.255.255.255 has less than 30 hops. Finally we've seen that 3 other 30+ hop traces occur in this data set and

manual inspection shows that these do not have a testbox as final entry. When following a similar approach for the traces with loops we find that only 12 traces have loops in them and no 255.255.255.255 addresses. Identifying such traces would be very useful for an ISP when troubleshooting the own network or in choosing an ISP to peer with. When many routing loops occur inside some other ISP's network this can be an indication for the reliability of this network.

One final visualisation of the entire data that is of interest is the combined graph of all six datasets. Only the semi-geographical layout is shown here, for the straight topological layout see the appendix. The graph shows the traces between the testboxes at 6 different moments in time. Therefore it is not a real representation of the network at any time. However, these visualisations do show that there are many more nodes than in the previous visualisations. If we look at the New-Zealand part (right bottom), for example we see that things have become a little more crowded. Of course this is a confirmation of what we've already seen in the initial analysis, by only looking at the numbers. This shows that when combining datasets of different time but with the same set of testboxes we find many alternative paths. This indicates the existence of dynamics in the routing between the testboxes.

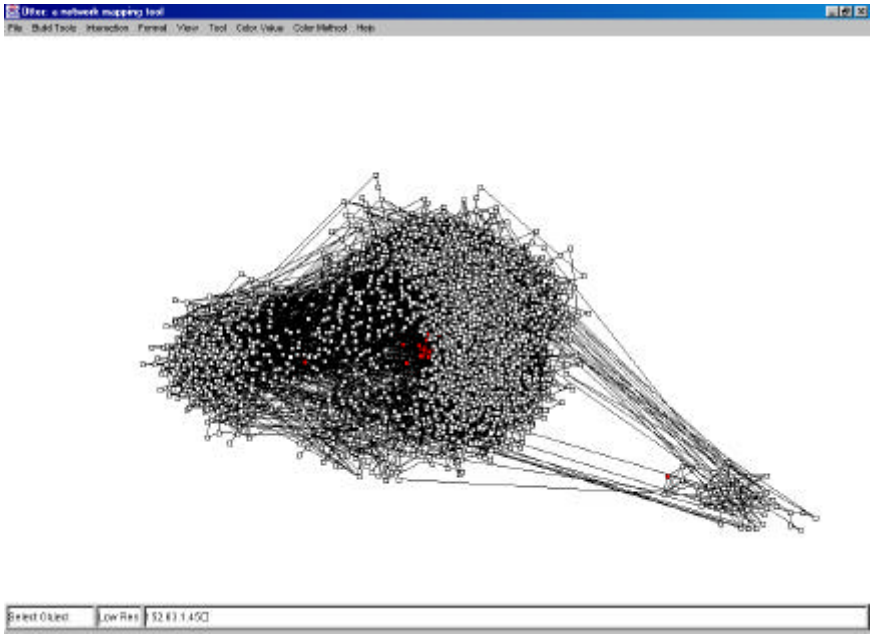


Figure 4-8. Visualisation of all tracevectors of the six datasets combined

We've seen in this paragraph that visualisation of large datasets is possible but at this level the statistics provides us with a little more insight than the actual visualisations, apart from the fact that the visualisations give insight in the complexity of the network. The next paragraph shows some other ways in which visualisation can be useful for especially ISPs or other network operators.

Practical use of visualisation

Apart from the visualisation of an entire dataset, and probably a lot more interesting is the visualisation of specific situations. This visualisation are especially valuable for testbox hosts or ISPs who want to improve their connectivity, troubleshoot their connections and determine smarter routes. There are many different possibilities, like using the visualisation to study the dynamic in the routing between two known endpoints (testboxes) and to troubleshoot a connection between those two points. Gaining insight in asymmetry of paths and their return paths is another possible use of visualisation. But also larger datasets could be studied, for example a visualisation of all the traces from and to one testbox at a specific moment in time shows the surroundings of this testbox and thus of the ISP that hosts this testbox. Again by repeating this visualisation at certain time intervals some insight in the stationarity of the routing to and from the testbox can be gained. Some of these visualisations will be made and explained in the remainder of this chapter.

One example of this is to study the traces between two testboxes at a certain moment in time. This gives the corresponding ISPs insight in how connectivity to each other is established. This is important if two ISPs or clients of ISPs have made arrangements about quality of service, for example.

Let's look at the connection between testbox tt01.ripe.net (Amsterdam) and tt11.ripe.net (Munich) at 01-07-2000 around midnight as an example of this, see Figure 4-9 for the visualisation of the tracevectors. The direction of the traces are found by manually inspecting the dataset and are

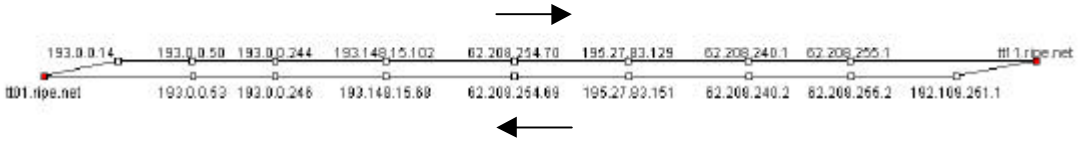


Figure 4-9. Tracevectors between testbox 1 and testbox 11 at 01-07-2000 around midnight

indicated by the arrows. At first it might seem as if the paths between the two testboxes are asymmetric. If we look more closely at the IP addresses, however, we see that there indeed is some symmetry. The nodes are arranged in such a way that nodes close together are probably part of the same communications subnet, because they share the same first numbers in their IP address. To illustrate this a small part of the physical connections between testbox 1 and testbox 11 is reconstructed in Figure 4-10. This approach shows some perspective for trying to map IP addresses onto physical devices, such as routers, which might be very helpful for ISPs in tracking and solving cross network problems, since not always the topology of adjacent networks is known. This also indicates that perhaps paths between two endpoints are not as asymmetric as is often suggested, although this was not extensively studied here. Note that in the Figure a straight line is drawn between the routers, where instead a cloud, indicating a single network that requires no routing to deliver internal packets could have been drawn. Also note that we can not be absolutely sure that the interfaces 193.0.0.14 and 193.0.0.53 belong to the same router, because another router could be present that is also connected to testbox tt01.ripe.net and to router B. However, it is likely that the configuration of routers is like in Figure 4-10. This shows the difficulties when trying to gain insight in real physical connections by using traceroute data.

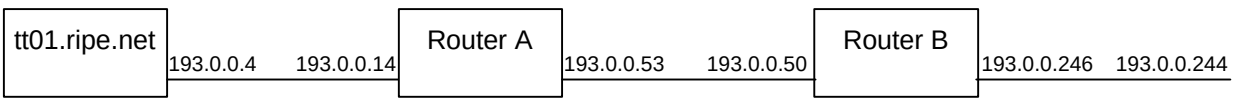


Figure 4-10. A reconstruction of a part of the connection between testbox 1 and testbox 11

When now looking at the same tracevectors one month later, we see that only two IP addresses differ from the ones found at 01-07-2000. Instead of 193.0.0.50 we find the entry 193.0.0.54, and instead of the entry 193.0.0.53 we find 193.0.0.49. This would suggest some dynamics in the routing of traffic between the two testboxes, so let's update our reconstruction a little bit, as shown in Figure 4-11. We see that it is highly likely that the IP addresses from 193.0.0.49 to 193.0.0.54 belong to the same communications subnet. Again, instead of drawing two separate lines, also one cloud could have been drawn, but in fact we can't be sure from this information that we do not actually have more separate lines between router A and router B. And of course the same remarks as in the explanation of Figure 4-10 holds, that we can not be absolutely sure of the actual router connectivity.

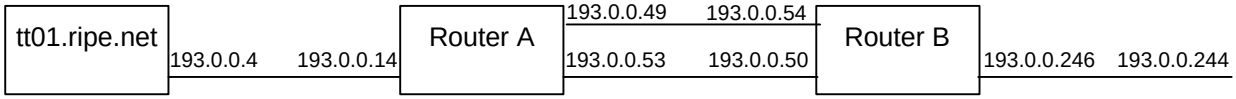


Figure 4-11. A reconstruction of a part of the connection between testbox 1 and testbox 11 (updated)

We learn several things from this small visualisation and analysis. First we see that although paths might seem asymmetric they are probably not. Second we notice that many IP addresses might map onto one router, which could be used to reduce the complexity of the visualisation, but the actual mapping of IP addresses onto routers can not be made with certainty. Finally, although different IP addresses are found in consecutive datasets, this doesn't always mean the route actually changes very drastically. In our example the route basically stays the same and merely some other interfaces of the same routers are chosen.

Our next visualisation is of all traces between testbox tt01.ripe.net and tt47.ripe.net found in the six different datasets. The resulting graph doesn't actually show an existing network, but it combines all information from the six datasets into on figure. At one time only two paths are actually used between the two testboxes. This visualisations merely shows that there is quite some dynamics in the paths between those two specific testboxes. Of course we do not know how often routes change, but it is clear that in a period of 6 months many changes in the routing occurred. A very simple rule of thumb can be derived from this, stating that the more complex the resulting graph is, the more routing changes have occurred. This can be easily seen by the fact that if all recorded tracevectors would be the same, the combined graph would be identical to the graph of any of the datasets. It has to be noted however, that reconstructing the actual physical, underlying graph would give much more insight in this matter of routing dynamics. Due to time limitations this is not done here.

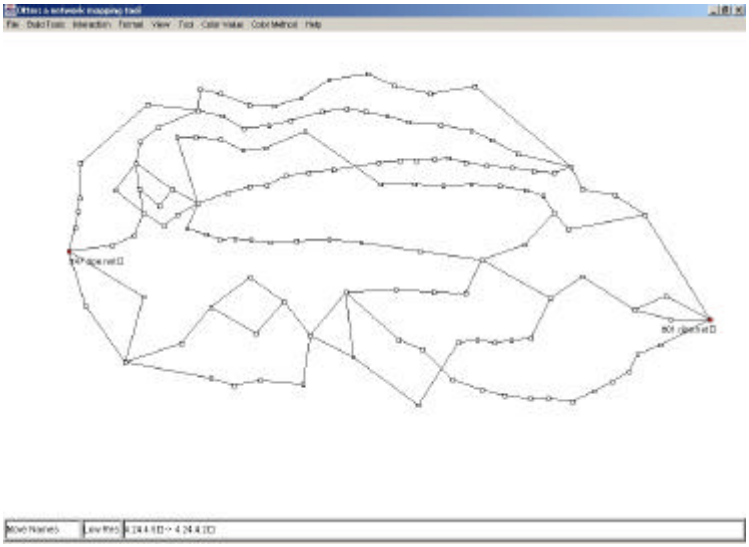


Figure 4-12. All traces between tt01.ripe.net and tt47.ripe.net

The previous example shows the connectivity between two testboxes, which is of interest if the two testbox hosts would like to study characteristics of this connectivity, in case they wish to reach or maintain some quality of service between them. However it might also be useful for a single ISP to obtain some knowledge of the way the other ISPs reach the own network. In this case a visualisation of all traces to the own testbox would be useful. The next Figure shows all traces to testbox tt01.ripe.net (blue node) from the 01-07-2000 dataset.

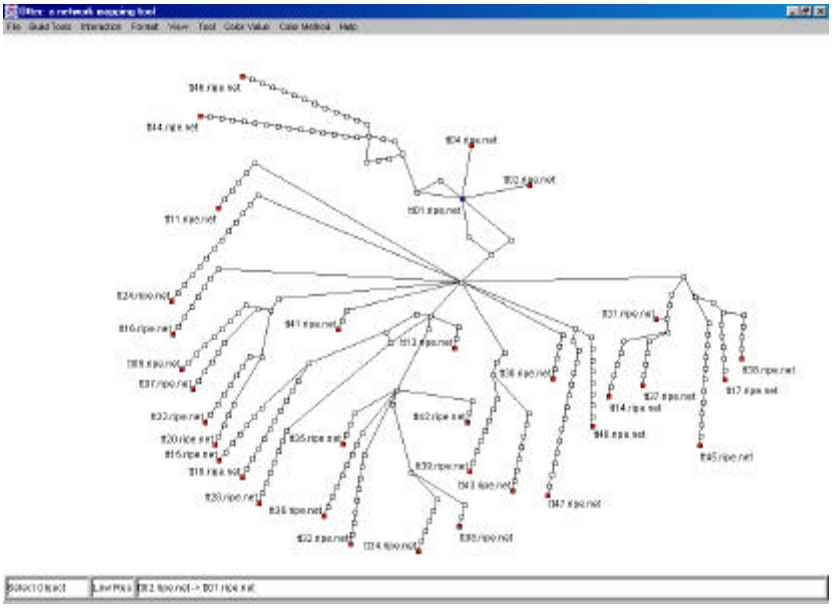


Figure 4-12. All traces to tt01.ripe.net on 01-07-2000

As we can see, most testboxes are “behind” one node close to tt01.ripe.net. This suggest that this ISP is highly dependent of this host for all incoming traffic. This, however can not be said with certainty, because we never know whether or not there are any additional connection that just didn’t show up in the traces. However if we where to detect these kind of “pseudo – dependencies” in many measurements, it might indicate that there indeed is some dependency on a certain host in the entire network. This would be very interesting to know, because such dependencies are not desirable.

Note that Figure 4-12 is manually manipulated to give a clear view of the nodes and names of the testboxes.

Finally the geographical layout of the graph build of all traces from and to testbox 1 at 01-07-2000 around midnight are shown in Figure 4-13. This graph does not really give much insight, although it is visually less complex than an entire dataset. For one ISP only this part of the network really matters, since this shows how their network is connected to the rest of the network.

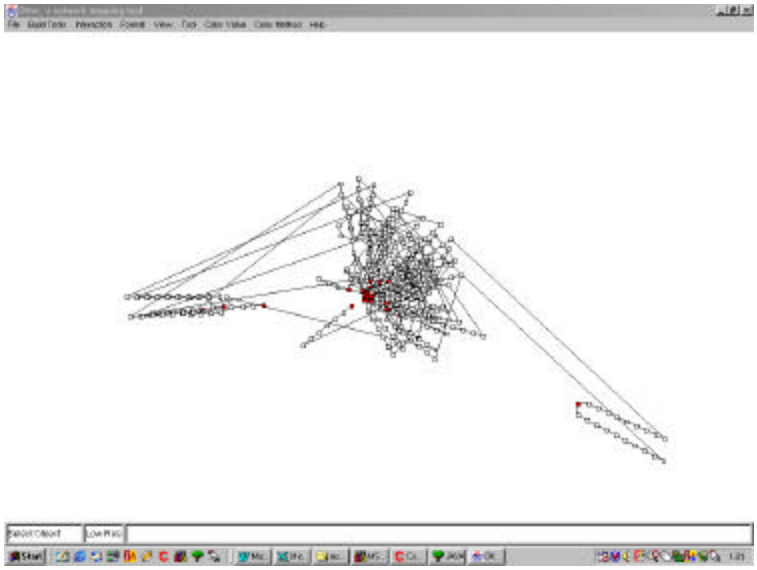


Figure 4-13. Visualisations of all traces to and from testbox 1 (20000701)

Many more visualisations could be shown here, but they would all illustrate the same point. Visualisation can be a powerful aid in analysing certain networking problems or configurations, it can provide useful insight in some special situations and it can help in understanding and supporting the modelling efforts within the UMEEPI project. However, this is not enough and additional information and insight is required to be able to give definite conclusions on certain problems, especially when looking at the level of IP addresses. Also some complexity reduction would be necessary to make such visualisation of the Internet more readable and thus more useful. At this time our visualisations are less informative than statistical analysis performed on the data, although they do illustrate how complex the Internet really is. One of the methods to reduce complexity was shown here, in Figures 5-10 and 5-11, where we tried to reconstruct the actual physical connectivity. The next chapter deals with another reduction method, providing us with a more high-level view of the network and giving us a major complexity reduction, especially visually. This is accomplished by mapping IP addresses to Autonomous System numbers.

5. Analysis and visualisation at the AS level

We've seen in Chapter 4 that our visualisations of the entire datasets lead to interesting, but far too complex graphs. Since we try to gain some insight in the topology of the Internet a solution to this problem needs to be brought up. Our initial analysis has shown that a great deal of complexity reduction would be achieved if we would be able to collect IP addresses of multiple interfaces that belong to one router into a single node. This, however is but one way to reduce the graph and we've seen that this also is not straightforward. We've seen in Chapter 3 that although the exact implementation and location of BGP speakers and border routers can be different, the Internet can be seen as a set of arbitrarily connected Autonomous Systems. We've learned that each router and thus each IP address is part of only one Autonomous System. Since there are far less Autonomous Systems than IP addresses, it can be easily seen that a major complexity reduction can be achieved by aggregating to this level.

Every host or router in the Internet belongs to an autonomous system, which can be referred to as a portion of a network, usually within the control of one organisation and usually running a single routing protocol. Routing between autonomous systems is done with a protocol that is defined as being an inter-domain or inter-AS protocol or an exterior gateway protocol [RP00]. Each AS has a unique number within the Internet.

The traceroute data from the RIPETT project consists of the subsequent IP addresses in a certain path. If these IP addresses can be clustered or mapped to AS-numbers a major simplification of the data can be made. This way we can draw a graph in which the nodes represent the different autonomous systems and the edges represent the possible paths between them. This will provide insight in inter AS connectivity and possibly in routing policies of different ISPs. On top of this, insight in the crucial role of certain backbones or internet exchanges can be gained. Also comparison of actual behaviour of routed traffic with routing policies articulated via BGP announcements are an area in which research is needed [CL99].

Being able to visualise the actually chosen paths between the Autonomous Systems provides us with a view of the actual traffic flows, which might be compared to routing policies announced through BGP, but also to those envisioned by the management of an AS. This way unintentional connections between Autonomous Systems can be detected or intelligent choices for peering contracts can be made. Furthermore, this level of detail is very interesting for operators of an autonomous system, because they can not influence the routing inside another autonomous system, but they can influence the path of autonomous systems taken to some destination. Finally these graphs can be very informative and interesting and aid in the effort within the UMEEPI project.

These considerations have led to the decision that being able to create a graph at the level of autonomous systems would be one of the main goals of this study. This chapter describes the process of obtaining these mappings and shows some interesting ways in which the resulting visualisation can be used.

Mapping IP addresses to AS numbers

The problem of reducing the IP graph to a graph at the level of autonomous systems starts with finding the corresponding AS-number with each IP address. In order to do this, we need to determine what source of information we can use. As we've seen in the chapter about routing in the Internet the BGP UPDATE messages contain the AS_Path attribute, which can indirectly show us what IP addresses "fall" within an AS. In order to obtain a mapping for all or most IP addresses encountered in the RIPETT data we would need to have access to multiple "higher level" BGP routers. The term higher level is used to indicate that the routers whose routing tables we would need to inspect need to have a complete and updated view of large portions of the entire network, which is more likely to occur at routers that at higher levels in the network. In other words we're talking about border routers of large backbones, which are difficult to get access to.

An example of a project that tries to obtain routing information in this way is the route-views project [DME01]. Here a router maintains BGP sessions with multiple large border routers of several participating ISP's, which results in a large database of routing information. This routing information could be used to derive mappings for the different IP addresses by looking at the AS_Path attributes of the corresponding routes and selecting the best-match AS from this data. This could be achieved by taking the last AS in the AS_Path, which reflects the originating AS. This, however still is not a guarantee that the correct mapping has been made. An example of this is the visualisation made at CAIDA, where the data provided by the skitter project [SKITTER] is combined with the route-views data and a graph of AS-connectivity has been constructed. A summary of this work can be found in at [CAIDA].

In fact this attempt is similar to our attempts to visualise the Internet topology at the level of Autonomous System except for the data used. The skitter project uses an approach where traces are performed from the testboxes to any destination, whereas the RIPETT project performs traces between the testboxes, thus obtaining a view of the portion of the network that connects the testbox hosts networks. This is a lot more useful for the specific ISPs than a high level graph of total Internet connectivity, which in turn is useful for academic research. As was said before, this thesis study focuses on visualisation at the AS level, but also on the practical use of this.

Although the route-views project looks promising and can give us the necessary BGP information, a different approach had been chosen for this project at a time when information about the route-views project was not yet available to us. Because major parts of the mapping program where already written at this time it was decided to stick with this approach and see how it would work out. The approach taken here was that the information about autonomous systems and the corresponding IP addresses would be extracted from the Internet Routing Registry, which will be explained in the next paragraph.

Internet Routing Registry

In this paragraph the concepts of the Internet Routing Registry are briefly addressed, only up to a level where a common understanding of the subject can be obtained, without getting into too much detail. The IRR in itself is a complex area and many subjects could not be studied thoroughly. However, enough insight was gained to be able to extract some useful information, namely the route objects with their origin AS, which will be explained later, from the IRR

The Internet Routing Registry (IRR) is a repository of routing policies[RFC2650]. Data from the Internet Routing Registry is free for use by anyone world-wide to help debug, configure and engineer Internet routing and addressing. At the IRR website it is stated that currently, the IRR provides the only mechanism for validating the contents of a BGP session or mapping an AS number to a list of networks [IRR]. This is exactly why we've chosen to use the IRR for obtaining the mappings, but as I've said the route-views project also looks very promising.

Currently there are various routing registry databases and a list of these can also be found at [IRR]. These databases together form the IRR, and in principle they complement each other. Many of these registries are public, like the RADB and RIPE registry and many others are private registries which contain the routing policies of the networks of the maintainers of the registry and their customers, like ALLTEL and WWNET. The registries maintain up-to-date copies of many of the other registries.

A specific language, RPSL, is used to register routing policies and configurations in the IRR. RPSL stands for Routing Policy Specification Language and is defined in [RFC2622]. RPSL is based on the ripe-181 specification, which is described in [RIPE-181]. This document also gives useful insight in this topic. RPSL constructs are expressed in one or more database "objects" which are registered in one of the registries as mentioned above [RFC2650]. Each of these database objects contain some routing policy information and some necessary administrative data. Its one of these objects that make the IRR interesting for our research.

The route object

Route objects in RPSL are used to define single (interdomain) routes from an autonomous system. In other words, the route objects give the routes from this AS which are distributed to peer ASs according to the rules of the routing policy, as explained in Chapter 3. To clarify, when a certain AS announces a certain prefix to a peer and when it is the originating AS, in principle this prefix should appear in the route field of a route object in the IRR. An explanation of the exact fields in a route object will follow shortly.

Another object has to be mentioned in this context, to show the difference with a route object. The aut-num objects describe propagation of routing information for an autonomous system as a whole whereas a route object defines a single route and thus contains all information regarding a routing announcement. See [RIPE223, RIPE181, RFC2622, RFC2650 and IRR] for more on the IRR. For us the syntax and information contained in a route object is sufficient, so lets look at an example of this.

```
route:   130.161.0.0/16
descr:   DUNET
origin:  AS1103
            remarks: This object is automatically converted from the RIPE181 registry
mnt-by:  AS1103-MNT
changed: ripe-dbm@ripe.net 19941121
changed: auto-rpsl@ripe.net 19991104
source:  RIPE
```

This example shows the different fields of a route object. Only two fields are of real importance to us. The route field or attribute is the primary lookup key and the origin attribute is the primary inverse key, which means that route objects can be searched by these two fields (along with some other non-primary fields).

The route attribute is the address prefix of the route and the origin attribute is the AS number of the AS that originates the route into the interAS routing system. As we've seen the route field contains a classless address. This address represents the exact route being injected into the routing mesh. In our example this is 130.161.0.0/16.

The value of the origin attribute is an AS reference like AS1103 in our example. It refers to an aut-num object and represents the AS injecting this route into the routing mesh. The aut-num object in turn can be referenced for the contact information for the corresponding AS [ripe 181] and thus for this specific route.

The importance to our study of these two fields becomes clear when realising that when, for example the address 130.161.23.13 is encountered in the dataset, this is most likely to be an address that "belongs" to AS1103, because this AS originates the route that contains this address. In principle, this provides us with a mapping for all encountered IP address. This can be understood by realising that the fact that they appear in a trace means that traffic can be routed to that IP, which in turn means that a routing announcement has been made for the corresponding prefix. This, finally suggest that an entry should be present in the IRR, which, however is not so straightforward as we will see in the next paragraph where the limitations of this approach are discussed.
The other fields shown here contain additional information concerning this route. For example the descr field contains a description, like DUNET, which is the Delft University NETwork, and the source field contains the specific IRR registry that this object is stored in. In our example this is the RIPE registry, but this could also have been RADB, ARIN or any of the other registries.

Whois

The protocol used to extract this information from these databases is the whois protocol. With a whois client, such as the one found at RIPE [WHOIS], it is possible to perform certain queries on a database that contains information on networks, autonomous system numbers (ASNs), network-related handles, and other related Points of Contact (POCs), such as the IRR databases. Although more powerful tools

are available in the RAToolSet [RATOOLS] for our research an whois client suffices to obtain the necessary information.

Without going into too much detail an whois client can be used to query a database using some command line parameters. The information required can be obtained by issuing commands such as:

```
whois –h whois.radb.net <network_IP>
whois –h whois.radb.net AS<Autonomous_System_Number>
```

The exact query performed and the further use of this data will be explained in the next paragraph.

Collecting the required information

Once it was decided that IRR data would be used, some approach had to be developed to extract the relevant information. This of course involves performing queries on the databases, but different approaches to this could be used. In essence there where two possibilities of obtaining the necessary data.

The first possibility was to query the databases at the time an IP address was encountered that needed to be mapped. In addition some sort of cache could be used to prevent multiple queries for one and the same IP address. This cache could even be used across multiple runs of the program, thus preventing having to query one IP address more than once. With this approach a query is based on a single IP address and thus a query would have to be performed many times for one dataset.

The second possibility under consideration was to first build some kind of table that maps the IP addresses on to AS numbers and that could be quickly accessed whenever a mapping was required. This would require the building of such a table only once and this would make the execution of the consecutive runs of the program much faster. However we also perform many queries that are in essence not needed, because many of the route objects queried to not contain any of the IP addresses encountered. Still, if this is only done once these time and speed considerations are less important.

First the appropriate data had to be collected and this was done using a whois query on the ARIN database. This database was chosen quite arbitrarily and instead the RIPE and RADB could have been used. The following query was used to obtain all route objects associated with one autonomous system:

```
whois -h rr.arin.net -T route  -i aut-num AS1103 >AS1103.txt
```

The whois client is asked to place a query at the rr.arin.net database (-h flag) and to search for all objects of type (-T flag) “route”, where a field of type “aut-num” exists with a value AS1103 (-i aut-num AS1103). Because some results can be quite long they are sent to a textfile (>AS1103.txt).
This way I collected all routes for the autonomous systems 1 through about 26000 that where available in the ARIN routing registry and stored them in individual 26000 textfiles. Initially the idea was to query for all 65535 possible AS number, but since all number above 22528 are not assigned to any autonomous system and since the querying process takes a long time it was decided to break of the queries at a certain point.
At this time we have the information we want, but not yet in a organised way that makes searching for IP addresses and finding a corresponding AS number easy. This functionality was added to the NodeLink 0.8 program, which will be described later. In short, this part of the program reads the 26000 files and creates a searchable table, a so called hashtable, and stores it for later use. This way the table has to be constructed only once from the 26000 files. Because all other parts of the program only work with this table, when a new source for the mappings is determined, only this part of the program has to be rewritten.

Limitations

Conceptually, a route object contains the information related to a routing announcement (BGP update message) which means that in principle the same information as in the core BGP routers and thus in the route-views project should be available through the IRR.

However, the information is inserted in the database by the network operators that choose to do so. This implies that not all network operators participate in the IRR project which may, of course compromise the reliability of the required information. Also due to this some routes that are indeed announced by certain ASs are not updated in the IRR or some routes that are in the IRR have already changed over time.

Also, although there can only be a single originating AS in each route object, sometimes a route is injected by more than one AS which leads to a situation where multiple routes with the same route field but with different origin fields occur in the IRR. This, of course, compromises our mapping, since an IP range can only be mapped onto one AS. This problem only occurs when two or more route objects have the exact same route field, for example 130.161.0.0/16.

The exact figures about how often these situations occur, what percentage of route-objects in the IRR are reliable and the exact implications of this in our mapping are not known.

NodeLink 0.8

In this paragraph the NodeLink 0.8 program is described. The updated and new parts of the code can be reviewed in appendix C. The first part of the program is not altered relative to NodeLink_0.7 and still functional in this version of the program. However, once the graph at the IP level is exported to an otter data file the program will create or read a stored hashtable, containing the mappings. First the creation of this table will be described.

Creating the hashtable

As we've seen we have 26000 files containing information about routes and autonomous systems. So the first thing to do is to parse all these files and filter out the relevant information. Since the autonomous system number could be derived from the filename of the specific result file, only route fields had to be found. Since all route objects start with the string "*rt:" or "route:" it is easy to filter out the relevant lines. Let's look at an example to clarify this:

```
% ARIN Internet Routing Registry Whois Interface

route:          193.52.100.0/24
descr:          UFR Sciences
                2 Rue de la Houssiniere
                44072 Nantes CEDEX 03
                FRANCE
origin:         AS1717
member-of:      RS-COMM_NSFNET
mnt-by:         MAINT-AS1717
changed:        nsfnet-admin@merit.edu 19931210
source:         RADB


route:          193.52.32.0/24
descr:          CRI Universite de Rennes1
                Campus de Beaulieu
                Avenue du General Leclerc,
                F-35042 Rennes CEDEX
                FRANCE
origin:         AS1717
member-of:      RS-COMM_NSFNET
mnt-by:         MAINT-AS1717
changed:        nsfnet-admin@merit.edu 19931210
source:         RADB
```

The above text is a fragment of the AS1717.txt file, which contains the results of the query for all route objects that contain an aut-num type field with a value of AS1717. All lines except the two lines that start with “route” are discarded.
The resulting lines are then parsed, which comes down to discarding the beginning of the line and storing the resulting IP range (route) as an “IPRange” object within the program. A table is created where the search key is the IP range and the value found is the AS number, see Table 5-1.

IPRange	As-number
…	…
193.52.100.0/24	1717
193.52.32.0/24	1717
…	…

Table 5-1. Example of the Hashtable

Before an IPRange is added to the table it is checked whether this IPRange is already present in the table, and if so, it is checked whether it “belongs” to the same AS. If it does, no update of the table is done, since the information found is already there. If this is not the case, the previous entry is discarded and the new IPRange is added to the table with the corresponding AS-number. This approach is chosen, because when two exact similar IPRanges are found, but two different AS-numbers are found it is impossible or at least very difficult to determine which of the two is valid, from the IRR data. Again, the focus was on creating practical results before correcting errors in the data. Still, this approach is rather rigorous and definitely would need improving in later versions of the program.

Note that this approach is only chosen when two exactly the same IP ranges are found. If, for example the IP range 130.161.0.0/16 is already present in the table and mapped to AS1103 and a new IP range of 130.161.23.0/24 would be found which “belongs” to AS1200 they are both stored in the table. This might seem like a corruption of the table since now, for example, the address 130.161.23.12 seems to be mapped to both AS1103 and AS1200. However, by always first searching in the table for the most specific match the IP range of 130.161.23.0/24 would always be found first and the 130.161.23.12 address would be mapped to AS1200. So the “thinking” is not done while storing the data, but while searching through the stored data.

Searching the hashtable

When the table is fully created from the whois data, it is stored as a whole, which makes a next run of the program a lot less time consuming. With the table and the table of unique hosts, which was stored while reading in the traced data, available, the final step of the mapping process can be performed. This can be best explained by using an example. Let’s return to our 130.161.23.12 address and assume that we wish to obtain the corresponding AS number.

First the hashtable is searched for an IPRange of 130.161.23.12/32, which in fact is the IP address itself. If this is not found the rightmost bit of the IPRange is set to 0 and the search is repeated for 130.161.23.12/31. Note that in this case the four parts of the IP address in this range stay the same, because the binary representation already ends with a zero. Next the right most two bits are set to zero and the search range 130.161.23.12/30 is searched in the hashtable. This step is repeated until the search range is of the form 130.161.23.0/8. Now when the table is searched, a mapping will be found. Finally the 130.161.23.12 address will be mapped to AS1200.

This approach makes sure that the most specific match will always be found first. If no mapping is found after 32 attempts (IP address is 32 bits) an AS number of 0 will be returned.

Constructing the AS graph

Finally, when all IP addresses are searched for in the hashtable, the reduced AS graph needs to be constructed. This is done in a similar way that the IP graph was constructed and the reader is

encouraged to look at the source code how this is implemented. One final point of interest is the way in which nodes that are not mapped are treated. The NodeLink 0.8 program provides two options, the first of which is to add all nodes to the resulting otter data file, which results in a graph containing both AS nodes and IP nodes. The second approach, is to leave out nodes that could not be mapped, so only AS nodes will be shown in the resulting Otter graph. Note that this differs from the approach in our previous version of the program where an entire trace with a “bad” node is removed from the graph. Here only the node that can not be mapped and its corresponding links are removed. This, however, breaks a trace in two parts and might cause nodes or parts of the graph to become separated from the rest of the graph.

Results

In the final paragraph of this chapter the results are discussed.

The hashtable

The first run of the program was intended to construct the hashtable from the IRR data collected in a previous step and save it to disk. Here the results of the mapping process are given:

```
Will now create a new ASHashtable:
Whois data directory (input)  = .\data\whois\
Working on file: 25850/25850
Total lines processed : 1850091
Total lines discarded : 1676734
Total IPRanges added  : 120783
Lines to be inspected : 2583
number of keys in Hashtable :120783
inconsistencies :23068
double entries :26923
Writing inspect vector to : inspect.txt
Writing inconsistent vector to : inconsistent.txt
Will now write ASHashtable to : .\data\hashtable.dat
```

A total of 25850 text files where processed, which contained a total of 1850091 lines, of which 1676734 did not contain any IP address, and thus no relevant information. A total of 120783 IP ranges where found and added to the hashtable. There where 2583 lines that contain some relevant information but for which a parser is not yet written. By looking at these lines and the information they contain some improvements to the parsing process and thus to the mapping process can definitely be gained. It happened 23068 times that an IP range that was already mapped to a certain AS was found in relation to another AS. This is 19% which means that there are many errors and inconsistent mappings in the resulting hashtable. This is one of the main limitations of this method, but as I’ve explained we chose to focus on gaining practical results and the improvement of the visualisation and the correction of errors is the next step in this incremental process. Finally it happened 26923 times that a specific mapping was found that was already present in the hashtable. When all files where processed, the hashtable was written to disk.

Also the list of lines that needed closer inspection and the list of inconsistent mappings where exported to a file. The first was meant for debugging and improving the parser and the following lines shows an example of the information held in this file:

```
AS1115.txt line: 40 : remarks:       replacement for Class-B: 138.117.0.0
AS114.txt line: 724 : *ho: 198.64.217.0/24
AS1200.txt line: 18 : remarks:       ias-int: 193.148.15.33 AS1103
AS1200.txt line: 19 :                ias-int: 193.148.15.34 AS1103
AS1200.txt line: 20 :                ias-int: 193.148.15.35 AS1128
AS1200.txt line: 21 :                ias-int: 193.148.15.36 AS1199
```

The filename and line number are given and then the string needs closer inspection is shown. Many of these lines contain some remarks that are not relevant to our mappings, like in the first line of the example shown here, which merely indicates that the IP range found was a replacement for a class-B

network, for that specific AS. The next line contain more interesting information. In the route object also an attribute called “holes” is defined as parts of the address space covered to which the originator does not provide connectivity. Let’s look at the whois results that contains this line:

```
*rt: 198.64.0.0/15
*de: SESQUI-CIDR-C
*or: AS114
*ho: 198.64.217.0/24
*ny: hostmaster@sesqui.net
*mb: SESQUINET-MAINT-MCI
*ch: sob@sesqui.net 971121
*so: CW
```

We see that autonomous system 114 originates the route to 198.64.0.0/15 but it does not provide connectivity to 198.64.217.0/24 which is a part of the 198.64.0.0/15 address space. This means that if we want to improve our mapping process, this has to be taken into account. At this time, nothing is done with this information and only the mapping of 198.64.0.0/15 will be found from this information. However, if 198.64.217.0/24 is found as a route originating in another AS, this will be updated in the hashtable and whenever an address has to be mapped that falls within this IP range the corresponding AS will be returned, because when searching, the most specific match will be found first. Still, if we want to improve our mapping, closer study has to be made of this subject.

Finally we see 4 lines that where found in the AS1200.txt file. This is the autonomous system of the Amsterdam Internet Exchange (AMS-IX), already explained in a previous chapter. When studying the whois query results we find that these remarks concern the route to 193.148.15.0/24. We see that a whole range of these remarks are made. As it turns out, these are the addresses of interfaces of border routers of the autonomous systems that are connected to the AMS-IX. Since they are connected through the 193.148.15.0/24 subnet, they must have an address that fall within this IP range, which has been allocated to the AMS-IX AS (1200). However, these routers and thus their interfaces are part of the connecting autonomous systems. Therefore each of these IP addresses actually belong to another autonomous system. This is what is made clear by these remarks and this is also important within our mapping process. Currently nothing is done with this information, because of time limitations. However, it also remains the question whether we want these addresses to be mapped onto their real AS number, or on the AS number of the AMS-IX. Probably if we where to map them to their real AS number, the AS 1200 would not appear in our graph, since only peering is done here, which means that two border routers of peering ASs are directly connected through the switches of the AMS-IX. Only traces that pass the AS1200 router (see Figure 3-13 in Chapter 3) would make the AMS-IX “show up” in our graphs. Therefore, perhaps mapping these addresses onto AS 1200 actually gives more insight in the situation. This is another reason why at this time nothing has been done about this.

Still our hashtable contains 120783 mappings of the form: 198.64.0.0/15 - AS114, which provides us with some means to perform the initial mapping and visualisation at the IP level. The remainder of this paragraph will show some interesting visualisations. Analogous to the previous chapter first some results of the entire datasets are shown, which are useful for general research purposes, after which some more practical uses of the visualisations are shown.

Visualisation of entire network

Now that we have the hashtable it is possible to reduce the IP graphs to AS graphs. Figure 5-1 show the 20000701 dataset. If we compare this to Figure 4-7 we find that a major complexity reduction was established. The central node in the graph (coloured red) is AS 1755 which is a major European autonomous system: the EBONE. We see that this node has great connectivity with other nodes and thus with other autonomous systems.

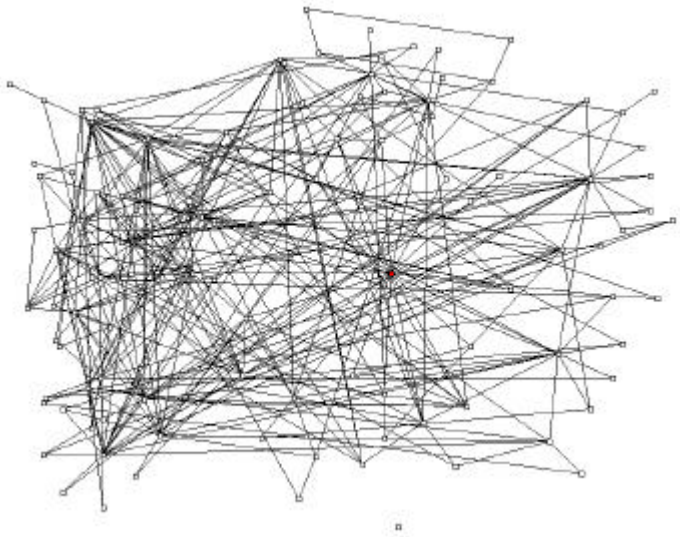


Figure 5-1. The AS level graph of the 20000701 dataset

One other thing of interest is the node that is isolated at the bottom of the graph, which is AS 3228. This is caused by our chosen method of dealing with unmapped IP addresses. As we've seen there are two options, one being to discard the IP nodes and leave these nodes and their connections out of the graph. This option was chosen here which resulted in one node becoming isolated. This is because the only connectivity of this node to the rest of the graph was provided by these unmapped nodes, so leaving them out isolated this node from the rest.

Appendix E shows a graph for each of the 6 datasets and the combined data, but all of them are similar in the sense that there are still about 80 to 100 nodes and 225 to 300 links. So, although a major complexity reduction has been gained, still the graphs remain fairly complex for visual inspection. However graphs with fewer nodes are faster to analyse using some computer tool and they can be a useful starting point for further study.

Table 5-2 shows summarises some numerical results from the NodeLink 0.8 program, such as data about the mapping process and on the number of nodes and links in the graphs before and after reduction. The last two columns show the number of nodes in the AS graph and it needs to be noted that these totals are taken when the unmapped nodes have been removed from the graph. The percentage of mapped and failed nodes are shown in brackets next to the absolute numbers.

Dataset	Nodes (IP)	Links (IP)	Mapped	Failed	Nodes (AS)	Links (AS)
20000701	2085	3104	2008 (96%)	77 (4%)	112	302
20000801	2103	3055	2028 (96%)	75 (4%)	108	298
20000901	1645	2371	1571 (96%)	74 (4%)	91	228
20001001	1528	2127	1437 (94%)	91 (6%)	85	225
20001101	1675	2405	1578 (94%)	97 (6%)	89	242
20001201	1556	2227	1462 (94%)	94 (6%)	90	216
Combined	3962	7365	3752 (95%)	210 (5%)	129	430

Table 5-2. Summary of some results from the NodeLink_0.8 program

We see that in all cases only a small fraction of the initial number of nodes and links remains, indicating a major complexity reduction. For the 20000701 dataset for example only 5% of the total number of nodes and 10% of the number of links remain. Similar values are found for the other five datasets, but the combined dataset even shows greater reduction; 3% of the nodes and 6% of the links remain.

This last point gives us some useful insight in the dynamics at the AS level, compared to the dynamics at the IP level. When we combine the 6 datasets we find approximate 4000 different, unique IP addresses, whereas each dataset alone contains about 2000 different, unique IP addresses. This in itself shows us that there is at least some dynamics at the IP level, because the IP addresses in the different datasets have to differ to be counted as unique IP addresses in the combined dataset.

However, when we look at the AS level we see that the average dataset contains about 100 nodes, where the combined dataset contains only 129 nodes. This indicates that in the subsequent datasets, only marginal changes in the encountered autonomous systems occur. This doesn't proof that the same routes are taken at the AS level, but still it indicates that the network between the testboxes consists of approximately the same autonomous systems over these six datasets.

The graph of the combined dataset is shown here for completeness and it can be compared to the graph at the IP level in Figure 5-2. The central node is still 1755 and the two isolated nodes are 8284 and 3228

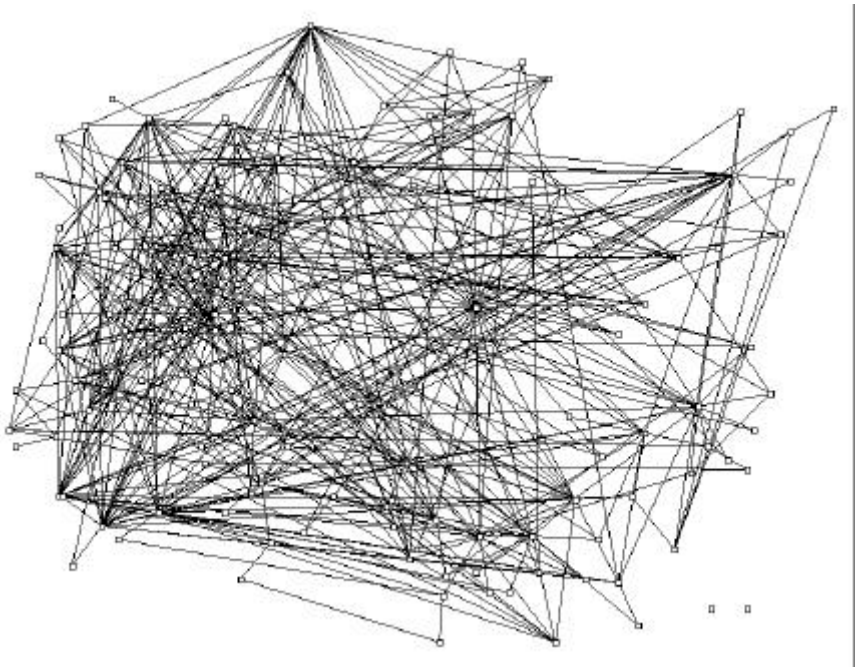


Figure 5-2. AS graph of the six combined datasets.

When we look at the graphs of Figure 5-1 and 5-2 we see that there are some nodes that have many connections, such as 1755 in the centre of the graph, which means that this is an highly connected autonomous system. In other words the 1755 node has a high degree. However there are also many nodes that have only one or two neighbours, the stub autonomous systems and the multihomed autonomous systems respectively. This calls for a closer examination of the degree of the nodes and how this is divided across all nodes. The histogram in Figure 5-3 gives an idea of this.

Here the number of nodes that have a certain degree are shown and we can see that there are many nodes that have 2, 3 or 4 connections with other autonomous systems, and there are few nodes that have more than, say 20 connections. In total 66% of all nodes have more than two connections. Also we see that there are four nodes that have no links at all, which is a consequence of our mapping method. Although these values do not accurately represent actual number of neighbours, they still give insight in the importance or connectedness of some of the autonomous systems.

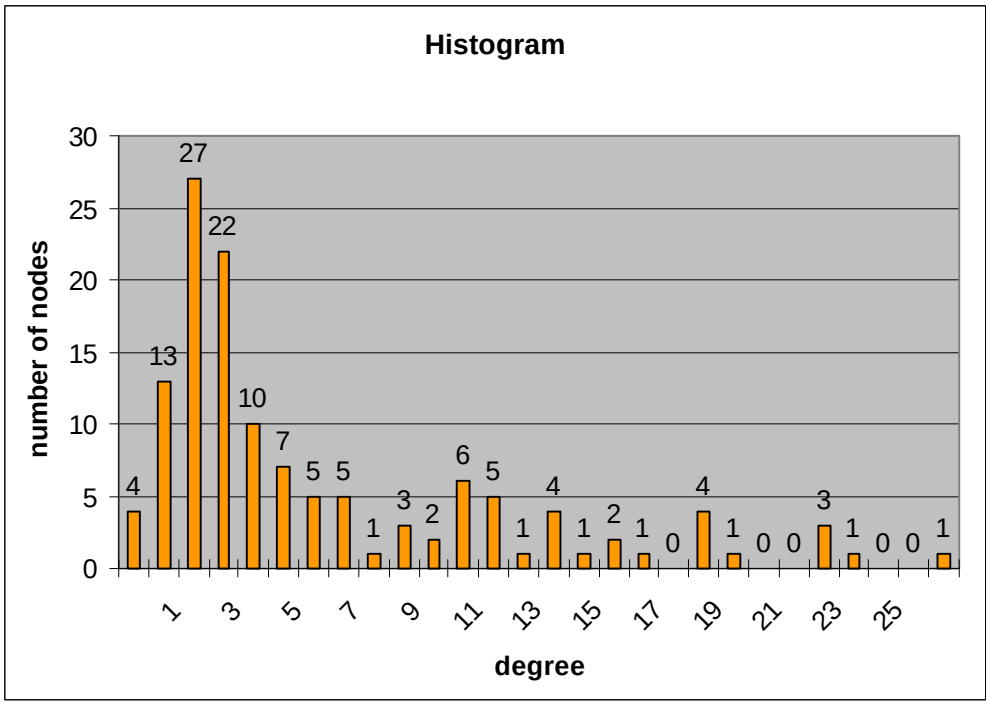


Figure 5-3. Histogram of the degree of the nodes versus the number of nodes with this degree

The autonomous systems that have many links in our graph, peer with many other autonomous systems and they provide transit service to these other autonomous systems as well, since they DO NOT host a testbox themselves and they DO appear in the tracevectors. This means that these autonomous systems are probably some of the major backbone providers or peering points and therefore it is interesting to study these a little more. Table 5-3 shows some additional information from the IRR about some of the more connected autonomous systems in our graph.

AS-number	Degree	Description
1755	27	Ebone Transit Core
3561	24	Cable & Wireless (MCI Backbone)
701	23	Advanced Networks and Services / UUNet
3320	23	Deutsche Telecom AG (German backbone)
5459	23	London Internet Exchange (LINX)
1	20	BBN Planet
1200	19	Amsterdam Internet Exchange (AMS-IX)
6453	19	Globe Internet, Canada
1790	19	Sprintlink (former NSFNET ?)
1673	19	Advanced Networks and Services Core Systems

Table 5-3. Some of the highly connected nodes from the combined graph

Practical use of visualisation

In this final paragraph of this chapter I show some of the more practical uses of visualisation, with the focus on usefulness for ISPs troubleshooting or configuring the network.

A first example of this would be to visualise the network between two providers or other network operators. These kinds of visualisations give insight in how the traffic is routed to the destination network, and if some characteristics, such as average delay, of the autonomous systems that are passed can be acquired, this will help in configuring the own border routers. Knowing what autonomous are traversed is a first step in these kinds of analysis.

First, let's look at the traces between testbox tt01.ripe.net and testbox tt11.ripe.net in the 20000701 dataset, which we have already studied at the IP level in the previous chapter. Our mapping process was able to find a corresponding AS number for all 18 IP nodes and the mappings that where found are shown in Table 5-4, where also some additional information on the three autonomous systems, resulting from whois queries, is included. Note that testbox 1 resides within autonomous system 3333 and testbox 11 within AS 1273.. Figure 5-4 gives the resulting AS level graph.

tt11.ripe.net : 1273	aut-num: AS3333
192.109.251.1 : 1273	as-name: RIPE-NCC-AS
62.208.255.2 : 1273	descr: RIPE NCC
62.208.240.2 : 1273	descr: European Regional Internet
195.27.83.151 : 1273	Registry
62.208.254.69 : 1273	
193.148.15.68 : 1200	aut-num: AS1200
193.0.0.246 : 3333	as-name: UNSPECIFIED
193.0.0.53 : 3333	descr: Amsterdam Internet Exchange
tt01.ripe.net : 3333	(AMS-IX)
193.0.0.14 : 3333	
193.0.0.50 : 3333	
193.0.0.244 : 3333	aut-num: AS1273
193.148.15.102 : 1200	as-name: ECRC
62.208.254.70 : 1273	descr: Cable & Wireless ECRC GmbH
195.27.83.129 : 1273	descr: Landsbergerstrasse 155
62.208.240.1 : 1273	descr: D-80687 Muenchen
62.208.255.1 : 1273	descr: Germany

Table 5-4. mappings and information on the found ASs

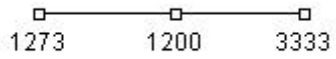


Figure 5-4. AS graph between tt1 and tt11

We've already seen that the trace most likely followed the same path in both directions at the IP level, but now we can confirm that at the AS level there is perfect symmetry in the paths in both directions. Traffic from testbox 1 to testbox 11 will go through AS1200 to AS1273 and traffic from testbox 11 to testbox 1 will go through AS1200 to AS3333.

Let's look a bit more closely at the IP addresses that are mapped to AS 1200. We find that 193.148.15.68 and 193.148.15.102 are the only two addresses that "belong" to AS1200. When we study the whois query results of AS1200 however, we find that these IP addresses are mentioned in the remarks of the route to 193.148.15.0/24 as explained in a previous paragraph. We've found the following two lines in these remarks:
ias-int: 193.148.15.68 AS3333
ias-int: 193.148.15.102 AS1273

This means that in fact AS3333 and AS1273 are peering at the Amsterdam Internet Exchange and the two border routers are connected through the 193.148.15.0/24 subnet, which is shown in Chapter 3, Figure 3-13. Therefore a different possible visualisation of the network between testbox 1 and testbox 11 at the AS level would be that shown in Figure 5-5 on the right. This option has not yet been build in the NodeLink_0.8 program, so this graph is drawn manually.

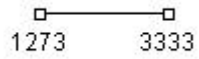


Figure 5-5. AS1200 removed

For completeness, when the trace from the 20000801 dataset is added, like we've done in the previous chapter in Figure 4-11 we still find the same graph at the AS level, which means that although the internal routing of AS3333 has slightly changed, the connectivity at the AS level remains the same for this trace in these two datasets.

Now let's look at the traces between testbox tt01.ripe.net and tt47.ripe.net from dataset 20000701, as an example of what happens when part of the mapping is not found. Testbox 47 is located in New Zealand. The first figure shows the visualisation of these traces on the IP level. The arrows indicate the direction of the measured tracevectors. Again we see many similarities between the IP addresses in both paths, indicating that roughly the same subnets are traversed.

There are 40 nodes in this graph and when we apply the mapping process to these nodes we find that only 36 of these can be actually mapped onto an AS. The four IP addresses that remain unmapped are: 209.0.227.1, 209.247.11.9, 209.247.2.1 and 63.211.89.1.

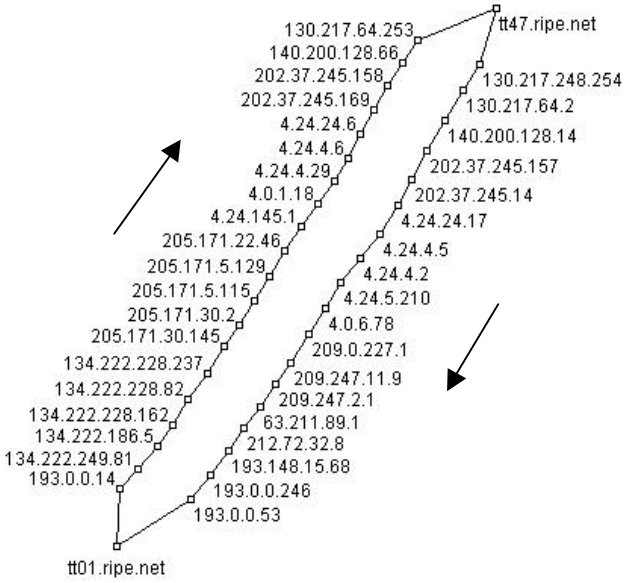


Figure 5-6. Tracevectors between tt01 and tt47 on 01-07-2000

The next two figures show the AS level graph; on the left with these four nodes still in the graph, and on the right with the unmapped nodes excluded. The autonomous system containing the testboxes are coloured red for clarity. Testbox tt01.ripe.net is located in AS3333 and testbox tt47.ripe.net is located in AS681.

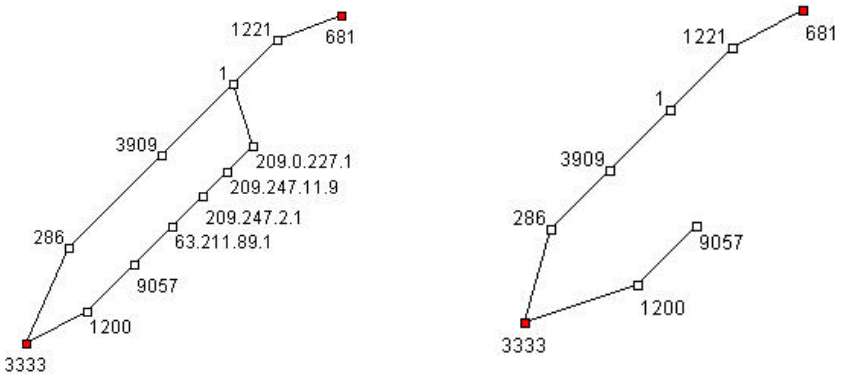


Figure 5-7. Two ways to deal with unmapped nodes.

These two graphs show the type of information that is lost by just leaving out the unmapped nodes. The left figure shows us that somehow the autonomous systems AS1 and AS9057 are connected, either directly or through one or more other autonomous systems. The right figure doesn't give us this information anymore and thus compromises our view of the real network.

One option is to manually try to obtain the mappings for the missing nodes. This of course is only possible for a small number of nodes, provided the necessary information is stored in some IRR. Since the RADB database mirrors many other IRRs this probably is a good starting point.
The following query was performed on the whois.radb.net database, with the following results:

```
D:\umeepi\tools\whois>whois -h whois.radb.net 209.0.227.1
route:        209.0.0.0/16
descr:        no more prtraceroute whiners !  Just kidding - we love you Nik.
origin:       AS3356
mnt-by:       LEVEL3-MNT
changed:      roy@Level3.net 20010518
source:       LEVEL3
```

This means the two 209.0.227.1 address maps to AS3356 according to these results. When repeating this method for the other addresses we find the following:

```
D:\umeepi\tools\whois>whois -h whois.radb.net 209.247.2.1
route:        209.244.0.0/14
descr:        no more prtraceroute whiners !  Just kidding - we love you Nik.
origin:       AS3356
mnt-by:       LEVEL3-MNT
changed:      roy@Level3.net 20010518
source:       LEVEL3
```

and

```
D:\umeepi\tools\whois>whois -h whois.radb.net 63.211.89.1
route:        63.208.0.0/13
descr:        no more prtraceroute whiners !  Just kidding - we love you Nik.
origin:       AS3356
mnt-by:       LEVEL3-MNT
changed:      roy@Level3.net 20010518
source:       LEVEL3
```

This means all these four unmapped nodes should actually be mapped onto the same autonomous system: AS3356. This also shows that our mapping process is still far from perfect, but time limitations prevent further updates to the program and thus to the hashtable used for the mapping. Simply querying more IRRs will probably make it possible to map nearly every IP address onto an AS number. However, this will probably also introduce more inconsistencies. Figure 5-8 shows the new graph after manually updating the Otter data file with the new found mappings.

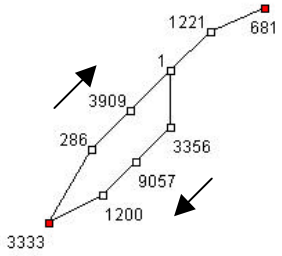


Figure 5-8. Manually updated graph

Now that we have the manually corrected graph of the traces between testbox 1 and testbox 47 we can focus on what we actually see. We see that the traceroute packets have traversed completely different autonomous systems in both directions, at least for one part of the route. The two different paths come together in AS1. There are many different possible explanations as to why traffic in one direction is routed differently from traffic in the other direction, all suggesting some kind of policy applied at a certain AS along the paths.

We can see what AS-paths are actually taken by our traceroute packets, and looking at graphs from the past and the present might give us more insight in routing policies at different autonomous systems. This, however has to be studied more thoroughly.

By visualising the tracevectors between two testboxes at different moments in time, we hope to gain insight in the dynamics in the chosen AS paths between two testboxes. The next series of pictures show the graphs of the traces between testbox tt01.ripe.net and tt47.ripe.net. Where needed the mappings have been manually improved in the way described above.

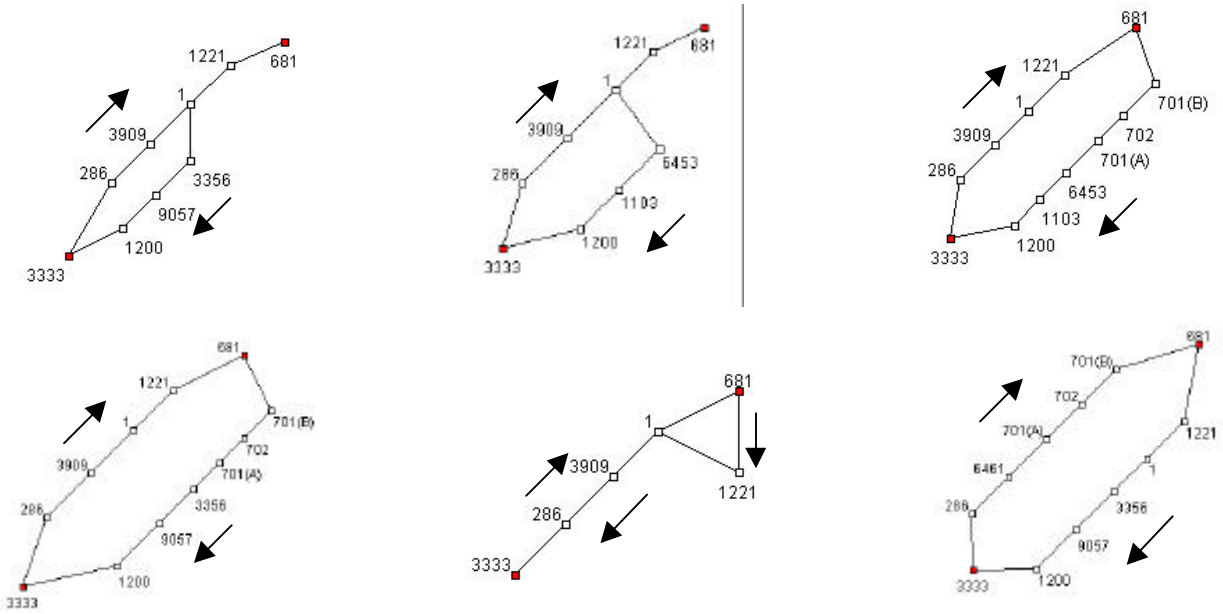


Figure 5-9. AS level graphs of tracevectors between testbox tt01 and testbox tt47 for all datasets

We see that in every dataset there is some difference in the chosen paths between testbox 1 and testbox 47. Most of the times the path from AS3333 to AS681 is more or less the same, but in the last dataset, we suddenly find a different path and even a new AS. This sequence of graphs shows, among other things, that although the nodes are more or less the same over time, the chosen paths vary at this timescale of 1 month between samples. It would be interesting to study these kind of dynamics on different timescales in order to gain more insight in routeflapping and other routing problems. Insight herein will also help us in our modelling effort. Note that in three of these graphs the autonomous system 701 is partitioned, which has been shown by adding a 701(A) node and a 701(B) node. This happens because in the corresponding tracevectors, first a few IP addresses are encountered that map to AS701, next an IP address is found that is maps to AS702, and finally some IP addresses that map to AS701 are found again. This means that traffic leaves AS701, passes AS702 and then enters AS701 again. This is called partitioning of an AS and has been briefly described in Chapter 3.

When we look at the combined graph in Figure 5-10 we see that we have a fairly simple graph even though almost every trace followed a different path. This is because the autonomous systems traversed remain more or less the same in all the datasets. This graph gives us some insight in the network between the two testboxes, which might be useful for configuring border routers at the testboxes autonomous systems. On the long run, these overall graphs might also give insight in routing policy and peering arrangements.

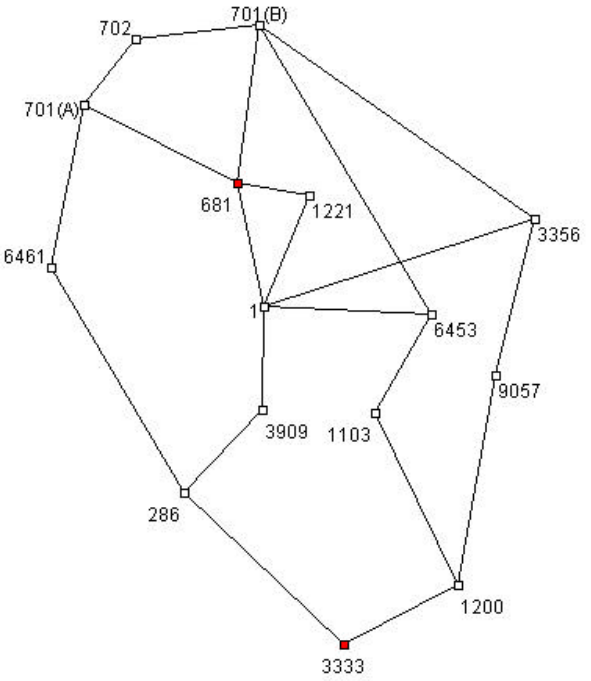


Figure 5-10. The combined graph

The following table shows some additional information on each of the autonomous systems in the graph. We see that AS9057 and AS3356 share the same description, and in fact they originate the same routes according to the IRR. However, the route objects corresponding to AS9057 have a much earlier date in the “changed” field, which suggests, that LEVEL3 has now been allocated AS3356, whereas it used to have AS9057. This would mean that AS9057 is in fact no longer used.

Aut-num	As-name / description /…	Aut-num	As-name / description
3333	RIPE-NCC-AS	9057	LEVEL 3 public IP network Europe
286	KPN-QWEST backbone AS	3356	LEVEL 3 public IP network Europe*
3909	WESTNET, Supernet Inc.	6453	GLOBE INTERNET, Canada
1	BBN Planet, GTE Internetworking	1221	TELSTRA, Australia
1200	AMS-IX	701	Alternet
1103	SURFnet	702	UUNet
6461	AS-ABOVE, AboveNet	681	New Zealand (part of NSFNet)

Table 5-5. IRR information on the ASs in the graph

One other use of visualisation is looking at all traces from and to a specific testbox, and visualising the “surroundings” of this testbox host. This can give similar insight as in the previous example, but this is not limited to the tracevectors that are measured between two testboxes.

When we recall Figure 4-13, where all traces from and to tt01.ripe.net on 01-07-2000 are visualised, we can see that this is a fairly complex graph. Originally there where 68 traces, but only 56 remained after using the three filters. This, however still leaves us with a graph of 436 nodes and 463 links. After our mapping process, however, 407 of the nodes could be mapped. After correcting the unmapped nodes and manually placing the nodes, we find the following graph of the surroundings of AS3333 (testbox 1) at 01-07-2000 around midnight. This graph contains 63 nodes and 68 links.

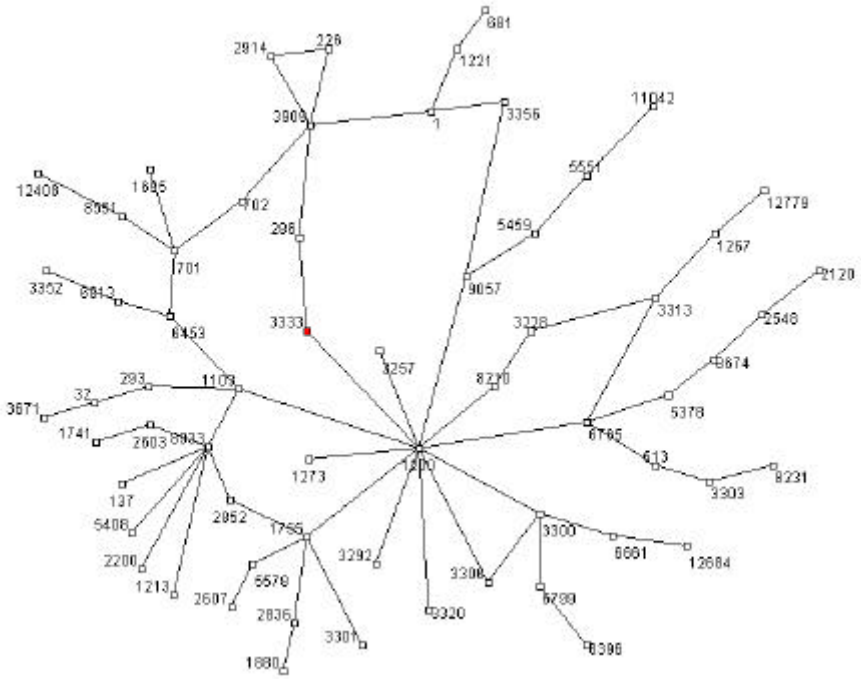


Figure 5-11. Surroundings of AS3333 on 01-07-2000

Note that AS3333 is the autonomous system of the RIPE-NCC where testbox 1 is located. There are some interesting observations we can make from these kind of visualisations. For a testbox host (ISP) it is interesting to know what ASs it is dependent of for connectivity to the rest of the Internet, except from some stub . In our case we see that AS3333 gets its global connectivity from AS1200(AMS-IX) and AS286(KPN-QWEST backbone). From this graph we can only tell that AS3333 is highly dependent of AS1200, since if this node is removed from the network, large portions of the network become unreachable. However, we are never certain there aren't any additional links that just did not show up in any of the tracevectors that where valid at that specific time.

So in order to gain some more insight in this we would need to look at the tracevectors over time. Figure 5-12 shows the tracevectors from and to testbox 1 combined from the six datasets.

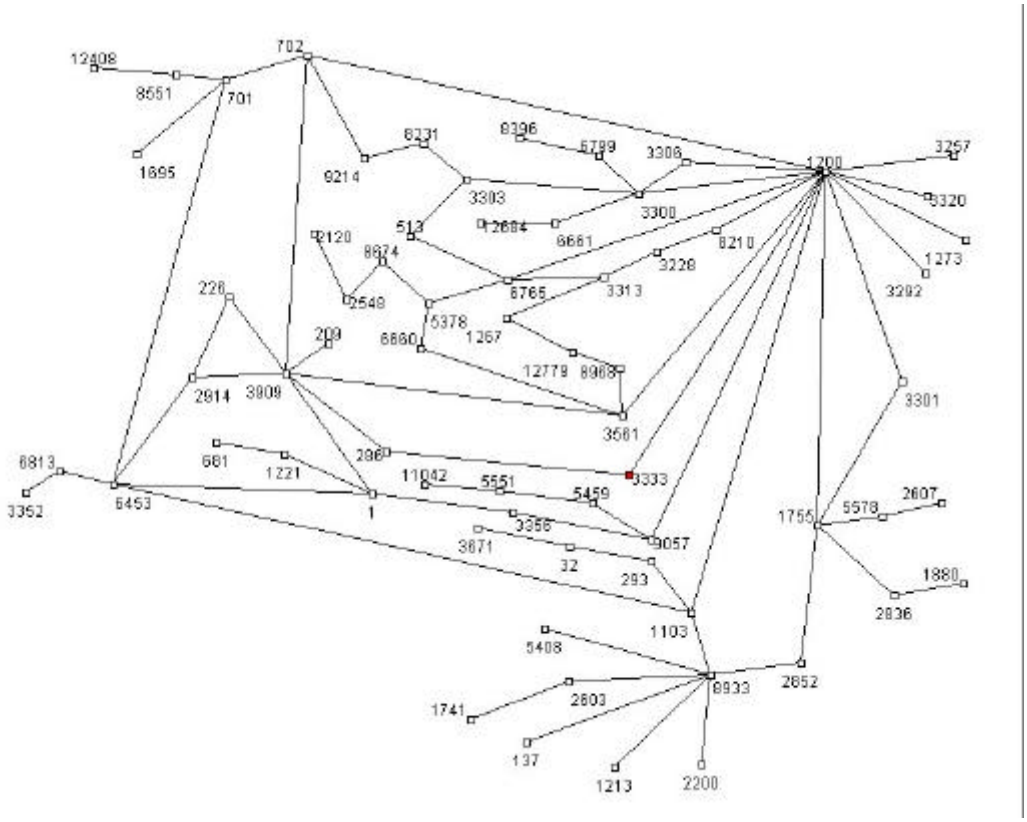


Figure 5-12. Surroundings of AS3333 based on 6 datasets

This graph shows us the surroundings of AS3333 based on several measurements taken at different moments in time. If we assume that the connections found between the different autonomous systems are not very dynamic, and that merely the chosen paths differentiate over time, than this gives quite a reliable view of the real network at the AS level. Since this graph is based on tracevectors to and from testbox 1 we can only say things related to AS3333. For example, we can see that AS3333 is still gets its connectivity of AS286 and AS1200, but we can also see that in case of failure of one of these, AS3333 remains fully connected. These kind of graphs are very useful in finding bottlenecks, dependencies. Furthermore, characteristics of the graph could be examined, and combined with the measured end-to-end delays these graphs can help in our modelling efforts.

We have seen in this chapter that complexity reduction can indeed be achieved by mapping IP addresses onto their AS-number and that the resulting graphs can be useful in many different way's. However, we have also seen that many problems and uncertainties remain, both in the mapping process and in the interpretation of the graphs.

6. Conclusions and recommendations

In this chapter the conclusions and recommendations are given.

Conclusions

The conclusions will be presented as answers to the research questions that where stated in the formulation of the problem in Chapter 1.

1. **How does inter-domain routing, or inter Autonomous Systems routing in the Internet work, both in practice as in theory?**
   A major goal of this thesis study was to increase our understanding of how the Internet and especially inter-domain routing works. The corresponding research subquestions have been answered, mostly in chapters 2 and 3:

   1.1. **How does the Internet work?**
   This question is extensively answered in the chapters 2 and 3, but in short the following description summarises this. The Internet is an arbitrarily connected set of routers and hosts, where the hosts are able to exchange IP packets and thus to communicate through the infrastructure of routers and links. It works based on the concept of packet-switching where the routers send packets closer to it destination, based on routing protocols, using routing information.

   1.2. **What are Autonomous Systems?**
   An autonomous system is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy.

   1.3. **How does routing in the Internet work?**
   Each router in the Internet knows about the IP addresses it can reach directly. This information is communicated to other routers using some routing protocol with its associated routing messages. Based on this information from its neighbours and possibly from all routers in the network a router constructs a forwarding table, stating which interface each incoming packet should be copied to in order to get it closer to its destination. This construction of the forwarding table is based on a routing algorithm such as the distance vector algorithm or the link state algorithm. For an extensive answer to this question refer to Chapter 3.

   1.4. **How does inter-domain routing in the Internet work?**
   Inter domain routing works similarly as intra-domain routing, only now inter-domain routing protocols are used for exchanging messages. In the case of the Internet the inter-domain routing protocol most used is BGP4, which is based on BGP messages containing network prefixes that the sending autonomous system provides access to. Again this is explained in Chapter 3 in great detail.

   1.5. **What is peering?**
   When two ASs decide that they will exchange traffic without using a transit AS, they make a peering agreement. In this case they have to have a direct physical connection. An example of peering is given in Chapter 3, Figure 3-13, where two fictive autonomous systems located at the Amsterdam Internet Exchange have a (private) peering relationship.

   1.6. **What is policy routing?**
   Policy routing is the ability to influence the routing information that will pass the border of an autonomous system. This in turn determines which traffic may pass this border. Note that there are other levels at which restriction to the traffic may be enforced, for example in firewalls, which are able to reject traffic from certain IP prefix ranges. Policy routing, however, operates at the level of autonomous systems. The policies are determined by the AS administration.

1.7. **How do peering and routing policy influence inter-domain routing?**
With more and more ASs deciding to make peering arrangements, the physical infrastructure of the network changes. Once two ASs have a peering relationship they can agree to only use this link for traffic originating or destining in the neighbours, but they also might agree to provide transit service to one another. This is done using routing policy. These kinds of arrangement make it less transparent what routes are taken and also inter-domain routing protocols, such as BGP4 have to contain mechanisms to support policy routing. It is clear that this makes the organisation of the Internet and inter-domain routing more complex.

2. **Is it possible to design and implement a tool that can map IP addresses to AS numbers and translates the given traceroute data into a topology map of the interconnected Autonomous Systems?**
From the first part of the study concerning interdomain routing we have learned that the Internet can be seen as a set of arbitrarily connected Autonomous Systems. We have learned that each router and thus each IP address is part of only one Autonomous System, which led to the conclusion that a major complexity reduction of the tracevectors and thus the visualisation hereof, would be achieved by mapping IP addresses onto AS number. This complexity reduction is necessary because graphs of the network at the IP level do not give the required insight. At the IP level statistics and maybe some numerical characteristics of the graph provide far more insight than a picture of the graph, although for small datasets, such as tracevectors between two testboxes, the IP level graphs can be useful, as is shown in Chapter 4.Still the concept of Autonomous Systems provides us with a useful aggregation possibility.

It is shown in Chapter 5 that it indeed is possible to visualise the network at the AS level and more over that this can provide us with some graphs with practical use. However, there are still many problems, such as dealing with erroneous data and obtaining mappings for all IP addresses. At this time we do not have a fully functional tool, but with manual inspection and alteration of the graph a lot of insight can already be gained.

2.1. **What information do we need to map IP addresses onto AS numbers?**
We have discovered that IRR data can provide the necessary information for the mapping of IP address onto AS number, by using the so called route objects, which contain information on network prefixes related to the autonomous systems that provides access to these network prefixes. However, the reliability of this data can be questioned, see answer to research question 2.3 for details on this.

Alternative and maybe more reliable (see question 2.3) sources of information might also become available. This could either be improved IRR data, that has been checked for inconsistencies and where every network operator participates or some other source, such as a high level view of the BGP data. The routeviews project by David Meijer will probably be very useful in this context [DME01].

2.2. **How can this mapping process be implemented?**
Although the source of information might not be as reliable as we really want, the concept of mapping IP addresses onto AS numbers and visualising the resulting graph has been proven in Chapter 5. By performing whois queries at several IRRs it is possible to build a table containing mappings for network prefixes onto AS numbers from the route objects. By searching this table from most specific to least specific, 95% of all addresses from the tracevectors can be mapped.

2.3. **What problems do we encounter when trying to map IP addresses onto AS numbers?**
First there are many inconsistencies (23068) in the sense that similar routes (or network prefixes) correspond with multiple autonomous systems. This is probably caused by the second problem concerning the reliability of the data. This is the fact that the IRR has to be manually updated by network operators which inevitable introduces human errors, combined with the fact that it is not obligated to update the information in the IRR. The third problem is that many route object have been entered in the IRR several years ago, which can be seen by the value of the “changed” field, while it is unlikely that nothing has changed since that time. This last reason might also explain why there are so many inconsistencies, because

maybe when new route objects are created the old route objects are not removed. This however has to be studied more thoroughly to be sure.

Despite of these doubts and inconsistencies in querying, we can still map a reasonable large part of the found IP addresses, about 95% of all addresses finds a mapping. Furthermore, when manually querying the unmapped IP addresses at some other IRRs, many additional mappings can be made. So the mapping process itself needs some improvements as well, which could be to query multiple IRRs instead of just one as is done here.

2.4. **How do we transform traceroute data into AS paths?**
This question is not really answered here, because in our prototype tool it was chosen not to convert the traceroute data into AS paths first. Instead an IP level graph is created, after which each node is mapped onto an AS number as described in Chapter 5 and above. All nodes that have the same AS number are compressed into one node, taking all links with other ASs along into the new node.

2.5. **What problems do we encounter in the RIPETT data?**
There are several limitations of the RIPETT data that prevent us from achieving high reliability for the accuracy of our AS level topology maps. Among these limitations, which are described in Chapter 4, are the limited measurement interval or scale, the facts that certain tracevectors are cut-off because they have more than 30 hops (see recommendations) and the occurrence of many unknown addresses inside the tracevectors, which are represented with the IP address 255.255.255.255.

Dealing with errors in the tracevectors is not very straightforward, but we have chosen not to focus on this part of the study, since Rob Meijer from KPN Research [ME01] is already doing research on this. We have learned, however that our approach of just disposing of tracevectors that have some errors might lead to major loss of data as is shown in Chapter 4. So either improvement of the measurements or further research in correcting errors or elegantly handling errors is needed, both of which are very complex and difficult tasks. One alternative of handling unknown hops is shown in Figure 4-6, but also here further study needs to show whether this is an acceptable way to deal with these errors.

2.6. **Does geographic placement of AS nodes give additional insight?**
The answer to this question depends on the scale of the visualisation. It is shown in Figure 3-6 that an Autonomous System is not necessarily geographically centred in one location. The Surfnet example shown here shows that an AS can span an entire country. Other ASs on the other hand might be located in one city or maybe even in one building. Therefore geographic placement of AS nodes is only useful when visualising ASs that have approximately the same scale, for example all major backbones might be visualised on a worldmap, or all national ASs on a map of a country. Note that this doesn't say anything about the usefulness of geographic information in determining physical distance and relating this to number of hops, as stated in the project proposal by prof. W.G. Vree [VR00]. This is a separate study that was not included in this research effort.

2.7. **Can a prototype visualisation tool be designed and implemented?**
It is shown in Chapters 4 and 5 that a tool that can visualise traceroute data at several aggregation levels can be implemented. Although there are still many problems to be solved it is shown here that AS level graphs can indeed be obtained and that some useful information can be derived from these.

3. **What is the use and relevance of topology maps at the AS level, what kind of information can be derived from these and what are the limitations and problems?**
The question of usefulness and relevance of the resulting AS graphs is answered in the following subquestions. Although many problems remained unsolved and new problems emerged, many new ideas where formed and a lot was learned during the process of programming a series of tools and analysing and visualising the data.

3.1. **How can AS level topology maps be interpreted, what do they represent?**
We have learned that many, especially smaller, Autonomous Systems are in fact the networks of one organisation or company. Larger Autonomous Systems are often collaborations of several organisations. An example of this is in Figure 3-6, where the Surfnet Autonomous Systems is shown. The most important thing we can conclude from this is that AS level topology maps show the relationships between ISPs and other network operators to some extend. Several characteristics, such as degree can be determined and this gives some indication of the connectivity of an organisation. Also insight into peering relationships and to some extend into policy might be gained.

Another aspect of AS level topology maps lies in routing. Since inter-domain routing protocols do not know how packets are routed inside Autonomous Systems, the routing process has to be based on information at the level of Autonomous Systems. The BGP routers also try to build a picture of the surrounding ASs and the prefixes they provide access to. It is not necessary for a router to obtain an AS level topology map of the entire Internet, but similar knowledge at a smaller scale is required by the BGP routers. At least they have to know who their peers are. So AS level topology maps represent the network that inter-domain routing protocols like BGP4 are working on. The fact that each node in this network is a network itself is not important for inter-domain routing, internal routing is done by some intra-domain routing protocol.

3.2. **How can AS level topology maps be helpful in understanding and modelling the Internet?**
Without visualisation several statistics can be determined and possibly modelled. For example being able to determine the average hop count is interesting for research groups that try to develop models to predict the hop count, because the real measured values could then be used for verifying the results of the models. But of course also this data itself could be used to determine the probability density function for the hopcount [MHH00]. This kind of research is interested in predicting the expected hop count, because it is believed that hop count and performance measures, such as end-to-end delay, are closely related. The predicted hop count could then be used in routing algorithms, to determine the best path to a destination. This, however is not straightforward and a study conducted by CAIDA [HFMN00] shows that there is little correlation between RTT (round trip time) values and the hop count, for most internet connectivity in the Asian Pacific part of the Internet.

Although this research used the RTT instead of the one-way delay, it is still clear that the number of hops traversed is only one of the many parameters involved in the performance of a certain network connection. If we were to base all routing decisions on the hop count we would only take into account the queuing delay and the processing delay at the routers. Quality of Service, however, is determined by many more parameters, most of which are unrelated to the number of hops traversed, such as transmission delays and propagation delays, jitter and the number of lost packets, which are all related to the type of medium or link used. If we have, for example, the choice between a three hop satellite connection (long propagation delay) or a seven hop connection over high speed links, a routing algorithm based on the hop count would lead to higher delays. So we can conclude that statistics such as the hop count will not provide the necessary insight and additional information is needed. We believe that this insight can be gained through visualisation in combination with statistical analysis. Visualisation is a necessary method in our modelling efforts

3.3. **How can AS level topology maps be helpful for individual ISPs?**
We can conclude that there are many practical uses of the data provided by the RIPETT project. We have shown, for example, that looking at a simple statistic like a 30 hop limit

might lead us to special situations that are very useful to discover. These kinds of analysis are important to testbox hosts, because it gives information on reachability and on configuration problems. We've seen an example of this with the tt28 testbox, which apparently was unreachable at midnight on 01-09-2000. Also by looking at this specific problem we found some indication of asymmetry in the tracevectors.

Also in routing, a tool that graphically displays traceroute information can be very useful for discovering potentially inefficiencies. Although tools like these already exist, the majority focuses on IP level visualisations, instead of, for example AS level visualisations. Identifying network topology at this level and gaining macroscopic insight into the Internet infrastructure can help in configuring the border-routers and determining where to peer with whom. This makes it possible for network operators to implement their own border-routers in such a way that both commercial (policies) and operational (delay, loss) demands are met.

Another area where insight in neighbouring Autonomous Systems might be useful is in security. The inter Autonomous System connections are essentially the gateways to the external world, with all potential threats that belong to that external world. By gaining insight in neighbouring ASs and optimising the number of peering points using AS level visualisation tools such as these, better Security strategies might be developed. Also determining where security problems occur can be aided by AS level visualisations. This subject, however has not been studied here.

Finally in network design and optimisations there are many chances for use of AS level topology maps. For example, the surrounding ASs can be studied and critical dependencies of other ASs can be found, and resolved. The way the surrounding ASs "see" the own network, an example of which is shown in Figure 4-12, is of importance if availability is an important Quality of Service measure. If all traffic to the outside world traverses one and the same AS, the own AS is highly dependent of the quality of this transit AS. In such a case an organisation should take measures because it is not preferable that the performance of another organisation (which provides transit service) dictates the performance of the own organisation. This could be solved by making peering arrangements with some other transit AS, or by arranging a Service Level Agreement between the own organisation and the transit provider. This also requires that performance has to be measurable, which in turn shows the importance of measurement projects such as RIPETT. This shows that important strategic decisions could be made if an accurate AS topology map of the surrounding ASs could be obtained, maybe even extended with AS and link characteristics. Our study has shown that acquiring these maps is indeed possible, although at this moment still too many uncertainties remain to base critical decisions on the resulting AS maps.

3.4. Can insight in dynamics of inter-domain routing tables be gained from the visualisations?

It is shown in Chapter 5 that some insight into dynamics can be gained. Merely by looking at a graph of all tracevectors over a period of time and comparing these to the graphs of all tracevectors at a specific moment of time we can see that different ASs appear in different graphs, without having to actually count the number of occurrences. Figure 5-2 shows an example of this, where about 130 nodes are found. Since all graphs of a specific moment in time have less than 100 nodes we can see that not all nodes that appear in the graphs at specific moments in time are the same, otherwise the overall graph would also contain about 100 nodes. This shows that there is at least some dynamics in the traversed Autonomous Systems.

In Figure 5-9 6 consecutive tracevectors between two testboxes are visualised and it is shown that each time different AS paths are taken. At this level it can be easily verified by eye that there is definitely dynamic routing behaviour. For larger datasets is should be possible to implement a program that determines whether paths are different, so indeed insight in dynamics can be gained, not only from visualisations, but also from the underlying graph representation (e.g. a matrix) and especially when these are compared at different time

intervals. However, the subject of dynamics is not explicitly studied here, and some recommendations are made at the end of this chapter in this light.

3.5. **Can we gain insight in routing policies and peering from the AS level visualisations?**
Some insight in general peering behaviour can be found. We have learned, for example, that many nodes (66%) in the resulting graphs have more than two connections (Figure 5-3), which means that many of the encountered ASs peer with more than two other ASs. This suggests that most the ASs are highly connected which gives them many routing alternatives. This also means that the dependencies on particular high level autonomous systems is probably not as high as would be expected, since there are many alternative routes available when this particular AS would fail. However, the autonomous systems with less connections is indeed dependent on the quality and reliability of some other autonomous systems. Visualisation on the AS level can give insight in these situations. Note that because many autonomous systems are highly interconnected, the resulting graphs still are reasonably complex, especially for larger datasets.

Also when studying certain specific Autonomous System using visualisation on different times it should be possible to build a detailed picture of its peering behaviour. Policy however, is a bit more complex too grasp using visualisations. We can make a distinction between transit and other Autonomous Systems, because all ASs that appear in the graph, that have more than one connection and that do not host a testbox themselves provide some transit service, otherwise they would not have showed up in the tracevectors. However, detailed insight in what AS accepts routing announcements and traffic from what other AS can only be gained if the visualisation tool would be expanded with directed links. We would also need to study the actual AS paths that where taken and not just the overall graph, because an AS might accept traffic from its peer, but not if that traffic originates in some other AS, for example. Still these expansions are possible and it therefore it should be possible to gain some insight in policy this way.

3.6. **Can we gain insight in the “Internet Backbone” from the AS level visualisations?**
It is indeed possible to determine what ASs are high level, or “backbone” transit providers, as is shown in Figure 5-3 and Table 5-3. Based on the degree of the nodes, which depicts the number of connections with other ASs, we could determine what ASs are most likely part of this global Internet backbone. Some of the ASs that where found are the Ebone Transit Core (AS1755), UUNet (AS701), but also Internet Exchange points such as the Amsterdam Internet Exchange (AS1200) and the London Internet Exchange (AS5459). Since it is known that these ASs are indeed high level, and also broadband networks, this shows that indeed nodes with a high degree are probably part of the “Internet Backbone”. Note that although visualisation can indicate what nodes have many links by visual inspection, some additional programming or analysis is needed to indeed calculate the degree of the nodes and determine what nodes are probably backbone nodes. Further study would be needed to improve the methods of determining the Internet Backbone and the possible use of this knowledge needs to be studied, although it is clear that by visualising the Internet Backbone with , say, another colour, or in some other distinctive matter, the resulting topology maps would become better readable and also many interesting uses might be possible, see the recommendations for more on this.

3.7. **What are the limitations of the developed tool and of the AS level topology maps?**
Many limitations have already been mentioned throughout the previous chapters. The most important ones are that the mapping process is far from perfect and the used data is also not optimal. Further more, many functionality that would greatly improve the usability of AS level topology maps has not yet been implemented or thoroughly studied. This in turn results in the fact that our AS level visualisations are still quite complex and far from optimal for use in our modelling efforts or for ISP. A lot of manual manipulation is needed to come up with maps that do give some insights. Of course also the errors in the RIPETT data present us with some problems, as was described above. See the recommendations for some suggestions on improvements.

3.8. **Is further research into AS level topology maps useful, or are other abstraction levels such as physical level topology maps more promising?**
We have shown that further research into AS level topology maps indeed is useful, and if some more reliable source of data for the mapping process can be found, the accuracy and the usability of the AS level visualisations will be promising. Many ideas for further study have been found, which shows that this subject is far from completed and that still a lot can be learned. Some of the ideas would be very helpful in our modelling efforts or for ISPs if they indeed could be realised. See the recommendations for some of these ideas.

Although it would be very interesting to obtain visualisations at the physical level, it is believed that this information is very hard, if not impossible to extract from trace vectors. This is because you just can not be sure whether or not two interfaces actually belong to one and the same router. Some assumptions can be made, but 100% certainty can not be gained, at least not at this moment. It might be possible to build a detailed map with the use of information from the different ISPs, however the current Internet consists of millions of hosts and above that, commercial ISPs are not willing to share all the details on their internal networks with the rest of the world. Therefor the only hope for obtaining physical level topology maps is that some measurement infrastructure or smart program can provide the mapping of IP addresses onto actual routers. Still, the fact that this is a great challenge alone makes it interesting to study this subject further. At this moment, however, study into other aggregation levels such as the AS level looks more promising.

We see that many of the research questions could be answered in a positive manner. This highly explorative study has led to some interesting conclusions, but also many subjects where only briefly addressed to and remain open ended. Some of these subjects appear in the next paragraph as recommendations, but some are only mentioned in the other chapters of this report. One final conclusion can therefor be that this subject is very complex and we are still far from developing highly useful tools and models that actually contribute to improvement of the end-to-end performance of the Internet. However, it is also shown that studying this subject further can aid in many different ways.

Recommendations

Many recommendations for further research, but also for practical use and improvement of the tools have been found. This final paragraph contains these recommendations.

First there are a series of recommendations for improving the developed tool:

- The whois queries should be performed on multiple IRRs, until the mapping is found. Only using one IRR site has proven not to be sufficient to obtain all mappings, while manually querying other locations did provide an answer.
- The parser of the whois results should be improved to account for anomalies such as holes and situations like with the AMS-IX, where a border router is part of another AS, but its interface on the AMS-IX network is mapped onto AS1200. It would be helpful if the user could choose to which AS he would like this IP address to be mapped to, depending on what the goals of the visualisations are.
- The unique hosts from a dataset are collected from all tracevectors. By counting the number of times a certain (interface of a) host is encountered in the tracevectors we would get some knowledge on important routers and moreover we could get insight in the type of graph (connectedness) we are dealing with by making an histogram as in Figure 5-3 for the AS level. Also calculating in-gress and out-gress of the nodes would provide useful insights.
- It would provide much insight if visualisation could take place as was shown in Figure 4-10. Here instead of the interfaces, the actual routers are nodes in the graph. First, this reduces the number of nodes and links significantly, and second this would give a more realistic view of the network under investigation.
- If the program would be rewritten so that it could query directly from the MySQL database used at the RIPETT project, the user would be able to make interactive selections at any timescales, which would largely improve the usefulness of the tool.

Next we have some recommendations on the graph layout itself and some analysis ideas for studying the dynamics in the chosen paths, for example. These are in essence also improvements to the tool, but they are shown here separately: Note that many more enhancements could be mentioned here, but the ones given here give some insight in possible further research.

- If each of the nodes gets its own specific location on some sort of a grid, visualising subsequent datasets could give more insight in the dynamics than with the present implementation. By visualising subsequent tracevectors between two testboxes with this improvement we could easily detect problems like routeflapping and probably gain more insight in routing policies.
- In addition paths that are travelled more often could get a different colour. When repeating this over longer periods of time, the links and nodes that carry the most traffic will eventually stand out above the less used links, giving some insight in the importance of certain autonomous systems and links. Also this could be used to determine the nodes and links that are essentially the core transit backbone, since they are responsible for most traffic in the network. This could be combined with the degree of the nodes and possibly other information on Internet Backbones. Eventually, If insight in the Internet Backbone could be gained, depicting this backbone in a different colour, for example, would enhance the visualisations and more over the analysis and modelling possibilities. It would be very interesting to study this subject further.
- Using directed graphs instead of undirected graphs would increase the usefulness of the tool in finding routing errors or in troubleshooting the network in some other way.
- A visualisation where the estimated delays of the links are shown next to the links would also be very useful, especially if the routers could obtain this information in real time and base their routing decisions on this. This requires accurate models for predicting the delay on the individual links, based on the end-to-end delay of multiple paths. Of course other characteristics such as loss and availability and so on could also be used inside the graphs. Further study into this is recommended.

Finally there are some additional recommendations:

- We have seen that mostly when the 30 hop limit is exceeded, this is caused by some error in the tracevector, which in turn might be caused by misconfiguration or failure of a router. Sending alerts to testbox hosts whenever this occurs might be useful in reacting timely on such events.
- Furthermore we have seen that some of the Intercontinental tracevectors are often 30 hops or longer. This is only a very small percentage of all traces, but for the related testbox hosts, (44 and 47) it is preferable that these tracevectors aren't corrupted. So, when looking at overall figures, the maximum TTL value of the used traceroute tool does not have to be increased, but when looking from the point of view of these two testbox hosts, it would be required to increase the maximum TTL value to provide better service to them. However, it has to be investigated further and at different timescales to be sure whether or not this is a real problem.
- This leads us to another recommendation for future research. Visualisations at different timescales and especially smaller timescales need to be made in the way described above, in order to gain insight in the dynamics of routing and in phenomenon such as route flapping. It should be possible to implement a tool that explicitly examines route changes in the data, at both the IP and the AS level. In this light it also has to be studied whether or not the time interval of measurement, once every 6 minutes, is small enough to detect al interesting events.
- Although erroneous tracevectors lead to loss of data, this is only a real problem for studying specific situations where we would explicitly like to know what paths are actually taken. On a higher level and when looking at entire datasets, correction of these errors becomes less important. Aside from information loss because of removing the erroneous tracevectors, we also miss data because we only perform traceroutes once every 6 minutes, or because we have only about 50 testboxes and not 2000 and so on. It would be better to improve the data collection method in some way and try to reduce the number of unknown entries in the dataset this way than to put a lot of effort in correcting the data afterwards. Therefore study has to be made how more accurate data, with less errors can be collected
- The same holds for the IRR data, used for the mapping of the hosts. Alternative sources, such as the routeviews project mentioned earlier, might provide us with more accurate and up to date information, which makes it less necessary to correct errors afterwards. This has to be studied.
- Effort should be put in making sure all testboxes are always working and working correctly. The different datasets often contained different subsets of the testboxes, which can be explained by the fact that sometimes testboxes are turned of, or some configuration error was made. This, however makes it more difficult to compare results and to study dynamics at different timescales.

This recommendation also comes from the idea that putting effort in collecting the right and valid data is better than correcting corrupted data afterwards.

This concludes our study into inter-domain routing and visualisation of tracevectors from the RIPETT data. Most of the goals have been met, but we have also learned that this is a very complex area and it is difficult to stay focussed on the actual goals during research. This leads to the final recommendation that in future research efforts, the scope of the study has to be very carefully determined.

References

[AMSIX] Topology map of Amsterdam Internet Exchange, http://www.ams-ix.net/About/ams-ix3.gif, last referenced August 2001

[BAR64] P. Baran, “On Distributed Communications Networks", *IEEE Trans. Comm. Systems,* 1964.

[CAIDA] An AS Internet graph by CAIDA, using skitter and routeviews data, http://www.caida.org/analysis/topology/as_core_network/AS_Network.xml, last referenced August 2001

[CIS01] Documentation department Cisco systems, Using the Border Gateway Protocol for Interdomain Routing“, Cisco Systems, 2001, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ics/icsbgp4.htm, last referenced August 2001

[CL00] K.C. Claffy (CAIDA), “Measuring the Internet”, IEEE Internet Computing Online, January 2000.

[CL99] K. C. Claffy, "Internet measurement and data analysis: topology, workload, performance and routing statistics", NAE '99 workshop, Co-operative Association for Internet Data Analysis (CAIDA)

[CS96] C. Semeria, “Understanding IP Addressing: Everything you ever wanted to know”, 1996, http://www.3com.com/solutions/en_US/ncs/501302.html, last referenced August 2001

[DHHLS98] P.P.A. van Dam, G. Hulst, H. van Hulst, Ir. H.A.M. Luiijf, Dr. M.E.M. Spruit, “Internet, intranet en beveiliging: het technisch kader.”, Praktijkreeks Informatiebeveiliging, NGI, 1998

[DME01] David Meijer, The Route Views project, http://www.antc.uoregon.edu/route-views/, last referenced August 2001

[FJ94] S. Floyd and V. Jacobsen, “The synchronisation of periodic routing messages”, IEEE/ACM Transactions on Networking, Vol.2, No 2, April 1994

[HFMN00] B. Huffaker, M. Fomenkov, D. Moore and E. Nemeth, “Measurements of the Internet Topology in the Asia-Pacific Region”, 2000, http://www.isoc.org/inet2000/cdproceedings/8e/8e_3.htm, last referenced August 2001

[HNC99] B. Huffaker, E. Nemeth and K. Claffy, “Otter: A General-Purpose Network Visualisation Tool”, CAIDA, 1999

[HO98] H. Uijterwaal, O. Kolkman, "Internet Delay Measurements using Test Traffic", proceedings of the 1998 NLUUG Spring Conference.

[HUI95] C. Huitema, “Routing in the Internet”, 1995

[IPPM] Internet Engineering Task Force, IP Performance Measurements Working Group (IETF IPPM-WG), http://www.ietf.org/html.charters/ippm-charter.html, last referenced August 2001

[IRR] The Internet Routing Registry Homepage, http://www.irr.net/docs/overview.html, last referenced August 2001

[ISOC00] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, Stephen Wolff , “A Brief History of the Internet, version 3.31”, Last revised 4 Aug 2000, http://www.isoc.org/internet-history/brief.html, last referenced August 2001

[ITO99] M. Ford, H.K. Lew, S. Spanier and T.Stevenson, ”Internetworking Technology Overview”, Cisco Systems, 1999, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm, last referenced August 2001

[KLE61] L. Kleinrock, "Information Flow in Large Communication Nets", RLE Quarterly Progress Report, 1961.

[KLE64] L. Kleinrock, “Communication Nets: Stochastic Message Flow and Delay”, 1964.

[KPN] KPN Research website, http://www.research.kpn.com/research/flash/_site_eng/index.html, last referenced August 2001

[KZ99] Sunil Kalidindi and Matthew J. Zekauskas, "Surveyor: An Infrastructure for Internet Performance Measurements", INET'99 conference, 1999.

[ME01] R. Meijer, presentation given at UMEEPI meeting, april 2001, TU Delft

[MHH00] v. Mieghem, Hooghiemstra, v.d. Hofstad, "A Scaling Law for the Hopcount, Delft University of Technology, ACM SIGCOM 2000.

[MI01] P v. Mieghem, preseantation given at UMEEPI meeting, april 2001, TU Delft

[MOCL96] T. Monk and K.C. Claffy, "A survey of Internet Statistics / Metrics Activities", 1996

[OH97] O. Kolkman, H. Uijterwaal, "Internet Delay Measurements using Test-Traffic, Design Note", RIPE-158, june 1997.

[PA94] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections", IEEE/ACM Transactions on Networking, 2(4), pp316-336, 1994

[PA97] V. Paxson, "Measurement and Analysis of End-to-End Internet Dynamics", Ph.D. Thesis, University of California, Berkeley, UCB-CSD-97-945, 1997.

[PA99] V. Paxson, "End-to-End Internet Packet Dynamics", IEEE/ACM Transactions on Networking, Vol.7, No.3, pp. 277-292, June 1999

[PAMM98] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, Framework for IP Performance Metrics, RFC 2330, Informational, May 1998.

[RATOOLS] The RA Tools Set, http://www.isi.edu/ra/RAToolSet/, last referenced August 2001

[RFC1771] Request for Comments 1771, “A Border Gateway Protocol 4 (BGP-4)”, 1995

[RFC1772] Request for Comments 1772, “Application of the Border Gateway Protocol in the Internet”, 1995

[RFC1918] Request for Comments 1918, “Address Allocation for Private Internets”, 1996

[RFC1930] Request for Comments 1930, “Guidelines for creation, selection, and registration of an Autonomous System (AS)”, 1996

[RFC2622] Request for Comments 2622, “Routing Policy Specification Language (RPSL), 1999

[RFC2650] Request for Comments 2650, “Using RPSL in Practice”, 1999

[RFC791] Request for Comments 791, “Internet Protocol”, 1981

[RFC792] Request for Comments 792, “Internet Control Message Protocol”, 1981

[RIPE] RIPE Network Coordination Center website, http://www.ripe.net/ripencc/about, last referenced August 2001

[RIPE-181] T. Bates, E. Gerich, L. Joncheray, J-M. Jouanigot, D. Karrenberg, M. Terpstra, J. Yu, “Representation of IP Routing Policies in a Routing Registry”, 1994

[RIPE-223] J.L.S. Damas, A. Robachevsky, “RIPE Database Reference Manual”, 2001-08-23

[RIPE37] Test Traffic Measurements, Status and Plans, presentation by Henk Uijterwaal at the RIPE 37-meeting, Amsterdam, 14 September 2000.

[RIPETT] RIPE Network Coordination Center: Test Traffic Measurements Project main page, http://www.ripe.net/ripencc/mem-services/ttm/, last referenced August 2001

[RIPE-tt-wg] RIPE Test Traffic Working Group, http://www.ripe.net/ripe/wg/tt/index.html, last referenced September 2001.

[RP00] R. Perlman, "Interconnections, Bridges, routers, switches and internetworking protocols", 2nd edition, 2000

[SAMSON] Samson Habte Tesfazgi, “End-to-end Internet performance modelling, A tool for modelling the end-to-end performance of the Internet”, Delft University of Technology, August 2001

[SKITTER] The Skitter tool, CAIDA, http://www.caida.org/tools/measurement/skitter/, last referenced August 2001

[ST98] J. W. Stewart, III, “BGP4: Inter-Domain Routing in the Internet, 1998

[STA96] W. Stallings, “Data and Computer Communications”, Fifth Edition, 1996

[TAN96] A. S. Tanenbaum, “Computer Networks”, 3rd edition, 1996

[UKKW98] Henk Uijterwaal, Daniel Karrenberg, Olaf Kolkman, Rene Wilhelm, "Internet Delay Measurements using Test traffic: First results", SANE98 Conference, Maastricht, the Netherlands, November 18-20, 1998.

[VR00] Prof. Dr. W. G. Vree, "Understanding and Modelling of End-to-End Performance of the Internet", research proposal, draft version, Department of Information and Communication Technology, Faculty of Technology, Policy and Management, 2000

[WHATIS] Whatis.com website, http://www.whatis.com, last referenced August 2001.

[WHOIS] The Ripe Whois Client, ftp://ftp.ripe.net/tools/ripe-whois-tools-2.4.tar.gz, last downloaded August 2001

[ZA01] Hobbes' Internet Timeline Copyright (c)1993-2001 by Robert H Zakon http://www.zakon.org/robert/internet/timeline/ , last referenced August 2001

[ZHPA00] Y. Zhang, V. Paxon, S. Schenker, “The stationarity of Internet Path Properties: Routing, Loss and Throughput”, 2000

Appendix A. A sample from dataset 20000701

962402400
tt01.ripe.net tt02.ripe.net 958559303 965778342 193.0.0.2
tt01.ripe.net tt04.ripe.net 959351596 965648213 193.0.0.6
tt01.ripe.net tt07.ripe.net 960544701 963262537 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.67 195.158.225.221
195.158.235.41 195.158.247.37 195.158.226.102 192.36.147.3 192.108.195.117 192.108.195.106 192.36.143.148
tt01.ripe.net tt08.ripe.net 962398203 962405505 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.34 255.255.255.255
255.255.255.255 255.255.255.255 255.255.255.255 255.255.255.255 255.255.255.255 193.10.252.21
tt01.ripe.net tt09.ripe.net 962402369 962402579 193.0.0.14 134.222.249.81 134.222.186.5 134.222.228.162 134.222.228.82
134.222.228.246 134.222.229.125 195.66.224.48 195.66.224.48 193.45.4.89 194.17.1.162 194.17.1.162 194.17.1.70
195.67.159.40
tt01.ripe.net tt11.ripe.net 962400199 962402484 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.102 62.208.254.70
195.27.83.129 62.208.240.1 62.208.255.1 192.109.251.32
tt01.ripe.net tt13.ripe.net 962400852 962408153 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.34 145.41.7.141 145.41.7.102
192.87.106.111
tt01.ripe.net tt14.ripe.net 962402024 962405946 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.92 195.89.1.106 195.89.0.65
195.89.0.74 192.65.185.137 164.128.33.205 164.128.33.33 164.128.35.34 164.128.33.106 164.128.141.4 164.128.138.66
tt01.ripe.net tt15.ripe.net 962357041 962454903 193.0.0.14 134.222.249.81 134.222.186.5 134.222.228.162 134.222.167.2
134.222.228.222 134.222.228.226 134.222.228.206 134.222.166.2 255.255.255.255 193.149.1.57 194.179.0.177
194.179.0.153 194.179.3.155
tt01.ripe.net tt16.ripe.net 962398960 962405014 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.104 195.249.2.129
194.239.225.169 195.249.4.165 195.249.112.229 195.249.12.246 195.249.96.237
tt01.ripe.net tt17.ripe.net 962397971 962403137 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.74 148.122.65.89
148.122.65.94 148.122.66.66 130.186.32.6 130.186.32.9 130.186.40.34 193.43.2.18
tt01.ripe.net tt19.ripe.net 962400048 962407251 193.0.0.14 134.222.249.81 134.222.186.5 134.222.228.162 134.222.167.2
134.222.228.222 134.222.228.34 205.171.24.113 205.171.4.70 146.188.162.210 146.188.163.66 146.188.138.118
146.188.178.181 146.188.177.129 157.130.252.222 192.115.106.246 10.20.1.18 10.20.1.14 212.116.160.49 192.115.187.100
tt01.ripe.net tt20.ripe.net 962401726 962403843 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.81
212.1.192.177 212.1.196.73 212.1.192.166 195.178.64.60
tt01.ripe.net tt22.ripe.net 962396626 962403340 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.67 195.158.225.221
195.158.235.41 195.158.235.34 213.174.70.42 213.174.70.38 213.174.70.46 195.158.242.10 195.158.242.214 62.168.97.106
194.160.23.2
tt01.ripe.net tt23.ripe.net 962382338 962542350 193.0.0.14 134.222.249.81 134.222.186.5 134.222.228.162 134.222.167.2
134.222.228.222 134.222.228.26 205.171.24.113 205.171.4.70 146.188.162.210 146.188.163.66 146.188.138.118
146.188.178.181 146.188.177.121 157.130.219.58 207.24.7.8
tt01.ripe.net tt24.ripe.net 962402052 962402467 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.44 195.90.65.105
195.90.65.125 195.90.65.134 195.206.64.50 195.206.64.206 195.170.0.6 195.170.5.46 212.70.194.2 212.70.192.6
212.70.195.130 212.70.195.122
tt01.ripe.net tt28.ripe.net 962393496 962407553 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.90
145.41.7.105 145.41.0.34 134.55.24.13 192.68.191.17 255.255.255.255 255.255.255.255 134.79.196.38
tt01.ripe.net tt30.ripe.net 962396332 962403048 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.44 195.206.67.130
195.206.67.242 194.154.198.198 212.56.224.2
tt01.ripe.net tt31.ripe.net 962400084 962407475 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.81
212.1.192.177 212.1.192.41 255.255.255.255 192.65.185.39
tt01.ripe.net tt32.ripe.net 961063953 963216326 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.81
212.1.192.177 212.1.192.89 212.1.196.26 193.206.134.62 193.206.132.146 130.192.53.1 130.192.2.9 130.192.227.253
130.192.68.200
tt01.ripe.net tt34.ripe.net 962401605 962404193 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.81
212.1.192.177 212.1.192.102 212.1.192.154 193.10.252.209 193.10.252.202 193.166.187.174 193.166.4.10 193.166.4.105
tt01.ripe.net tt35.ripe.net 962399432 962413069 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.81
212.1.192.177 212.1.192.106 212.1.192.254 193.1.198.10
tt01.ripe.net tt36.ripe.net 962401865 962402557 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.34 145.41.7.141 145.41.0.81
212.1.192.177 255.255.255.255 255.255.255.255 255.255.255.255 255.255.255.255 255.255.255.255 194.199.224.113
194.199.224.42 193.54.184.129 193.54.185.123 129.88.9.1
tt01.ripe.net tt37.ripe.net 962401790 962407357 193.0.0.14 193.0.0.54 193.0.0.244 193.148.15.44 195.90.65.105
195.90.64.218 195.206.65.50 195.90.64.178 255.255.255.255 255.255.255.255 195.112.71.35
tt01.ripe.net tt38.ripe.net 962402374 962403451 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.92 195.89.1.105 213.38.248.2
195.89.0.186 194.177.110.9 192.94.212.129 192.106.1.131 193.70.143.74 213.254.0.6
tt01.ripe.net tt39.ripe.net 962396743 962406808 193.0.0.14 193.0.0.50 193.0.0.244 193.148.15.111 194.25.6.153
62.156.140.125 194.25.121.173 62.156.139.242 194.25.120.209 194.25.120.194 212.185.8.114 62.156.134.161
62.157.150.125 62.157.150.94
tt01.ripe.net tt40.ripe.net 962383654 962848040 193.0.0.14 134.222.249.81 134.222.186.5 134.222.228.162 134.222.167.2
134.222.228.222 134.222.229.222 205.171.24.113 205.171.4.238 144.232.14.13 144.232.14.42 144.232.184.70 208.17.177.33
255.255.255.255 212.5.128.15

Appendix B. The NodeLink_0.7 program (11 classes)

```
package nodelink;

import java.io.*;
import java.util.*;

public class NodeLink{
  //Default file locations:
  static String testboxFile = ".\\data\\testboxes.txt";
  static String traceDir = ".\\tracefiles\\";
  static String dataDir = ".\\data\\";
  //variables
  static TestBoxes testboxes;
  static Traces traces;

  public static void main(String[] args){
    //first read in testboxes from file
    System.out.println("Will first read the testboxes from: " +testboxFile);
    testboxes = new TestBoxes(new File(testboxFile));
    //next collect the traces
    System.out.println("\nEnter filename of inputfile(will look in " +traceDir +"): ");
    String filename = traceDir + DialogTools.getInput();
    traces = new Traces(filename);
    traces.stats();
    System.out.print("\nFilter out traces that contain loops? (y/other)");
    boolean loop = DialogTools.getAnswer();
    System.out.print("Filter out traces that contain unknown hops? (y/other)");
    boolean unknown = DialogTools.getAnswer();
    System.out.print("Filter out traces where last hop is no testbox? (y/other)");
    boolean testbox = DialogTools.getAnswer();
    if (loop | unknown | testbox){
      traces.filter(loop, unknown, testbox);
      traces.stats();
    }
    //Now create a new Graph of the traceVector
    System.out.println("\nWill now create graph from trace data");
    Graph ipGraph = new Graph(traces);
    System.out.println("\nWill now export Graph to ODF file");
    System.out.print("Enter filename (will place in "+dataDir+"):");
    String file = DialogTools.getInput();
    ipGraph.exportToOdf(dataDir+file);
  }
}
```

```
/** this class defines the link objects. A link instance has some variables,
  * the two nodes that are connected by the edge, and a boolean
  * direction indicator, where true means bi-directional and false is uni-
  * directional, meaning from the first to the second node
  */
package nodelink;

import java.io.*;
import java.util.*;

public class Link {
  //instance variables
  boolean bidirectional;
  Node node1;
  Node node2;

  //constructor
  public Link(Node node1, Node node2){
    this.bidirectional = false;
    this.node1 = node1;
    this.node2 = node2;
  }
  //constructor
  public Link(boolean bidirectional, Node node1, Node node2){
    this.bidirectional = bidirectional;
    this.node1 = node1;
    this.node2 = node2;
  }

  public String toString() {
    return (node1 +" " +node2);
  }

  public void setBidirectional(boolean bidirectional){
    this.bidirectional = bidirectional;
  }
```

```
  public boolean equals(Object obj) {
    if (!(obj instanceof Link)){
      return false;
    }
    return (((Link)obj).node1.equals(node1))&(((Link)obj).node2.equals(node2));
  }

  public int hashCode(){
    return (node1.hashCode() + node2.hashCode());
  }
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class Node implements Serializable {
  String name;
  float latitude;
  float longitude;
  //for providing a location on the grid in Otter, currently not used
  //int x;
  //int y;

  public Node(Host host){
    this.name = host.name;
    this.latitude = host.latitude;
    this.longitude = host.longitude;
  }

  public boolean hasGeo(){
    return (!(latitude ==0 && longitude ==0));
  }

  public String toOtterGeoString(){
    return String.valueOf(latitude) +" " +String.valueOf(longitude)+" " +name;
  }

  public String toOtterString(){
    return name;
  }

  public Node(Hop hop){
    this.name = hop.host.name;
    this.latitude = hop.host.latitude;
    this.longitude = hop.host.longitude;
  }

  public String toString(){
    return name;
  }

  public boolean equals(Object obj) {
    //first check if it is a Node
    if (!(obj instanceof Node)){
      return false;
    }
    else {
      //two Nodes are equal if their names are the same
      return ((Node)obj).name.equals(name);
    }
  }

   public int hashCode(){
    return name.hashCode();
  }

}
```

```
/** this class defines a Graph object. A Graph consists of a Vector of nodes
  * and a Vector of links. Among its methods are exportToODF and reduce
  */

package nodelink;

import java.io.*;
import java.util.*;

public class Graph {
  //instance variables
  static Vector nodes;
  static Vector links;
```

```
//constructor
public Graph(){
  nodes = new Vector();
  links = new Vector();
}

//constructor
public Graph(Vector nodes, Vector links){
  this.nodes = nodes;
  this.links = links;
}

//constructor based on a traces object
public Graph (Traces traces){
  nodes = new Vector();
  links = new Vector();
  //first collect all the unique nodes, which are in the traces.hosts Vector.
  System.out.print("collecting nodes: ");
  int nodecounter = 1;
  int size = traces.hosts.size();
  for (int i=0;i<traces.hosts.size();i++){
    String progress = nodecounter +"/" +size;
    System.out.print(progress);
    Node node = new Node((Host)traces.hosts.get(i));
    nodes.add(node);
    nodecounter++;
    for (int q = 0; q<progress.length();q++){
      System.out.print("\b");
    }
  }
  //next get the links
  System.out.print("\ncollecting links from trace: ");
  size = traces.traceVector.size();
  int counter = 1;
  for (int i = 0; i<traces.traceVector.size(); i++) {
    String progress = counter +"/" +size;
    System.out.print(progress);
    //Take every trace in the traceVector one by one
    Trace currentTrace = (Trace)traces.traceVector.get(i);
    for (int j = 0 ; j<(currentTrace.hops.size()-1);j++){
      Hop hop1 = (Hop)currentTrace.hops.get(j);
      Hop hop2 = (Hop)currentTrace.hops.get(j+1);
      Node node1 = new Node((Host)traces.hosts.get(traces.hosts.indexOf(hop1.host)));
      Node node2 = new Node((Host)traces.hosts.get(traces.hosts.indexOf(hop2.host)));
      addLink(node1, node2);
    } //end of this hopvector
    counter++;
    for (int q = 0; q<progress.length();q++){
      System.out.print("\b");
    }
  }  //end of this traceVector
  System.out.println("\nDONE!         ");
  System.out.println("Nodes in graph             : "+nodes.size());
  System.out.println("Links in graph             : "+links.size());
}

public boolean addLink(Node node1, Node node2){
  if (node1.equals(node2)){
    //System.out.println("Link A-A type");
    return false;
  }
  else {
    Link link = new Link(node1,node2);
    if (links.contains(link)){
      //System.out.println("Link already in links!");
      return false;
    }
    else {
      Link reverse = new Link(node2, node1);
      if (links.contains(reverse)){
        //System.out.println("Reverse link already in links. Set bidirectional true!");
        ((Link)links.get(links.indexOf(reverse))).setBidirectional(true);
        return false;
      }
      else {
        links.add(link);
        return true;
      }
    }
  }
}

public void exportToOdf(String filename){
```

```
    //first create a new "otterlinks" Vector
    Vector otterlinks = new Vector();
    //then examine each link
    for (int i = 0; i<links.size(); i++){
      //get the positions of the nodes of the link in the nodes vector
      Link link = (Link)links.get(i);
      String pos1 = Integer.toString(nodes.indexOf(link.node1));
      String pos2 = Integer.toString(nodes.indexOf(link.node2));
      //create a string with the two positions
      //Todo, something with bidirectional link!
      String otterlink = pos1 +" "+pos2;
      otterlinks.add(otterlink);
    }
    //next write the odf file
    System.out.println("Writing ODF file:");
    try {
      File f = new File (filename);
      PrintWriter p = new PrintWriter(new FileWriter(f));
      p.println("t " +nodes.size());
      p.println("T " +otterlinks.size());
      //first the nodes
      System.out.print("Writing nodes: ");
      for (int i = 0; i<nodes.size();i++){
        String progress = (i+1) +"/" +nodes.size();
        System.out.print(progress);
        Node currentNode = (Node)nodes.get(i);
        if (currentNode.hasGeo()){
          p.println("N " +i +" " +currentNode.toOtterGeoString());
        }
        else {
          p.println("? " +i +" " +currentNode.toOtterString());
        }
        for (int q = 0; q<progress.length();q++){
          System.out.print("\b");
        }

      }
      System.out.println(nodes.size() +" written!       ");
      //then the links
      System.out.print("Writing links: ");
      for (int i = 0; i<otterlinks.size();i++){
        String progress1 = (i+1) +"/" +otterlinks.size();
        System.out.print(progress1);
        p.println("l " +i +" " +otterlinks.get(i).toString());
        for (int q = 0; q<progress1.length();q++){
          System.out.print("\b");
        }
      }
      System.out.println(links.size() +" written!       ");
      p.close();
      System.out.println("DONE!");
    }
    catch (IOException e){
      System.out.println("[ERROR]");
      System.out.println(e);
      System.exit(1);
    }
  }
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class Hop implements Serializable {
  Host host;

  //constructor
  public Hop(IP ip, Vector hosts){
    if (hosts.contains(new Host(ip))){
      host = (Host)hosts.get(hosts.indexOf(new Host(ip)));
    }
    else if (NodeLink.testboxes.containsIP(ip)){
      host = (Host)NodeLink.testboxes.getByIP(ip);
      hosts.add(host);
    }
    else {
      host = new Host(ip);
      hosts.add(host);
    }
  }

  public String toString(){
```

```
    return host.toString();
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof Hop)){
      return false;
    }
    else {
      return ((Hop)obj).host.equals(host);
    }
  }

  public int hashCode(){
    return host.hashCode();
  }

  public boolean isTestbox(){
    return (host.name.startsWith("tt"));
  }

  public boolean isUnknown(){
    return (host.ip.equals(new IP("255.255.255.255")));
  }
}
```

```
/** this class defines the IP objects. An IP instance contains
  * the four parts of a dotted-decimal notation IP address
  * and some useful conversion methods
  */

package nodelink;

import java.io.*;
import java.util.*;


public class IP implements Serializable{
  //instance variables
  public int one;
  public int two;
  public int three;
  public int four;

  //constructor (creates an IP instance, given the four parts)
  public IP (int one, int two, int three, int four){
    this.one = one;
    this.two = two;
    this.three = three;
    this.four = four;
  }

  //constructor (creates an IP instance, given a String)
  public IP (String s){
    int first = s.indexOf(".");
    int middle = s.indexOf(".",first+1);
    int last = s.lastIndexOf(".");
    this.one = Integer.parseInt(s.substring(0,first));
    this.two = Integer.parseInt(s.substring(first+1,middle));
    this.three = Integer.parseInt(s.substring(middle+1,last));
    this.four = Integer.parseInt(s.substring(last+1));
  }

  //constructor (creates an IP instance, given a long)
  public IP (long iplong){
    long rest = 0;
    this.one = (int)(iplong/Math.pow(2,24));
    rest = iplong - (long)(one*Math.pow(2,24));
    this.two = (int) (rest/Math.pow(2,16));
    rest = rest - (long)(two*Math.pow(2,16));
    this.three = (int) (rest/Math.pow(2,8));
    rest = rest - (long)(three*Math.pow(2,8));
    this.four = (int) rest;
  }

  //this method returns the decimal value of an IP object
  public long toDec (){
    long iplong;
    iplong = (long)(one*Math.pow(2,24)+two*Math.pow(2,16)+three*Math.pow(2,8)+four);
    return iplong;
  }

  //returns the string representation of the IP object
  public String toString (){
    return one +"." +two +"." +three +"." +four;
```

```
  }

  //this method overrides the equals method, stating that two IPs
  //are the same if the values of their respective ip variables are the same
  public boolean equals(Object obj) {
    if (!(obj instanceof IP)){
      return false;
    }
    return (((IP)obj).one == one) && (((IP)obj).two == two)&&
           (((IP)obj).three == three)&& (((IP)obj).four == four);
  }

  //this method overrides the hashCode method, which is mandatory if the
  //equals method is overridden
  public int hashCode(){
    return (int)this.toDec();
  }
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class Host implements Serializable {
  String name;
  IP ip;
  int asn;       //default = 0;
  float latitude;
  float longitude;
  String description;

  public Host(){
  }

  public Host(IP ip){
    this.ip = ip;
    this.name = ip.toString();
    this.asn = 0;
    this.latitude = 0;
    this.longitude = 0;
    this.description =null;
  }

  public Host(String name, IP ip, float latitude, float longitude, String description){
    this.name = name;
    this.ip = ip;
    this.latitude = latitude;
    this.longitude = longitude;
    this.description = description;
    this.asn = 0;
  }

  public String toString(){
    return name;
  }

  public boolean equals(Object obj) {
    return ((Host)obj).ip.equals(ip);
  }

  public int hashCode(){
    return name.hashCode();
  }

  public boolean isMapped(){
    return !(asn == 0);
  }

  public int getAsn(){
    return asn;
  }

  public void setAsn(int asn){
    this.asn = asn;
  }

  public boolean sameASN(Host other){
    return (other.asn == asn);
  }

  //public ... getMapping(){
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class DialogTools{
   public static String getInput(){
    BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
    String returnval = "" ;
    try{
      returnval = console.readLine();
    }
    catch (IOException e){
      System.out.println("Error getting user input. Goodbye!");
      System.exit(1);
    }
    return returnval;
  }

  public static boolean getAnswer(){
    BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
    String response = "" ;
    try{
      response = console.readLine();
    }
    catch (IOException e){
      System.out.println("Error getting user input. Goodbye!");
      System.exit(1);
    }
    if(response.startsWith("y")){
      return true;
    }
    else {
      return false;
    }
  }
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class Traces implements Serializable{
  public Vector traceVector; //contains the trace objects
  public Vector hosts;       //contains all the unique hosts in these traces

  //constructor
  public Traces (String filename){
    traceVector = new Vector();
    hosts = new Vector();
    System.out.print("\nReading from " +filename );
    File f = new File (filename);
    try {
      BufferedReader in = new BufferedReader(new FileReader(f));
      String line;
      //parse the file line by line
      int counter = 1;
      System.out.print(" :line: ");
      while((line = in.readLine()) != null) {
        String progress = "" +counter;
        System.out.print(progress);
        counter++;
        StringTokenizer sttk = new StringTokenizer(line);
        int tokens = sttk.countTokens();
        if(!(tokens<5)){
          //only lines with 5 tokens or more can contain a trace and will be parsed
          Trace trace = new Trace(line, hosts);
          traceVector.add(trace);
        }
        for (int q = 0; q<progress.length();q++){
          System.out.print("\b");
        }
      }
      System.out.println("DONE");
      in.close();
      //count();
    }
    catch (IOException e){
      System.out.println("Error parsing traces from file: " +filename +"! Goodbye!");
      System.exit(1);
    }
  }
```

```
public String toString(){
  return
  " traceVector: " +traceVector.toString();
}

//this method counts the total number of hops and prints some statistics
public void stats(){
  int totalHops = 0;
  float averageHops = 0;
  System.out.print("Collecting statistics : ");
  int size = traceVector.size();
  int counter = 1;
  //just check the hops of every traceVector one by one (testboxes are also here!)
  for (int i = 0; i<traceVector.size(); i++){
    String progress = counter +"/" +size +" : Total Hops found= " +totalHops;
    System.out.print(progress);
    counter++;
    Trace currentTrace = (Trace)traceVector.get(i);
    for (int j = 0; j<currentTrace.hops.size();j++){
      totalHops++;
      Hop currentHop = (Hop)currentTrace.hops.get(j);
    }
    totalHops--; //every trace has one hop to many in hop vector (first = testbox)
    for (int q = 0; q<progress.length();q++){
      System.out.print("\b");
    }
  }
  averageHops = (float)totalHops/traceVector.size();
  System.out.println("Done                                                                                ");
  System.out.println("Traces in traceVector          :" +traceVector.size());
  System.out.println("Total Hops                     :" +totalHops);
  System.out.println("Average Hops                   :" +averageHops);
  System.out.println("Unique hosts (incl. Testboxes) :" +hosts.size());
}

public void filter(boolean loop, boolean unknown, boolean testbox){
  int loops = 0;
  int unknowns = 0;
  int testboxs = 0;
  System.out.print("Filtering: ");
  int size = traceVector.size();
  int counter = 1;
  String progress = "";
  for (int i = 0; i<traceVector.size(); i++){
    for (int q = 0; q<progress.length();q++){
      System.out.print("\b");
    }
    progress = counter +"/" +size;
    System.out.print(progress);
    counter++;
    Trace currentTrace = (Trace)traceVector.get(i);
    if (testbox&&(currentTrace.lastIsNotTestBox())){
      testboxs++;
      traceVector.remove(i);
      i--;
      continue;
    }
    if (loop&&(currentTrace.containsLoop())){
      traceVector.remove(i);
      loops++;
      i--;
      continue;
    }
    if (unknown&&(currentTrace.containsUnknown())){
      traceVector.remove(i);
      unknowns ++;
      i--;
      continue;
    }
  }
  for (int q = 0; q<progress.length();q++){
    System.out.print("\b");
  }
  System.out.println("DONE         ");
  System.out.println("\nTraces with loops removed: " +loops);
  System.out.println("Traces with unknown hops removed: " +unknowns);
  System.out.println("Traces with last hop no testbox removed: " +testboxs);
  recollect();
}

public void recollect(){
  hosts.clear();
  System.out.print("Recollecting unique hosts: ");
  int size = traceVector.size();
```

```
    int counter = 1;
    //just check the hops of every traceVector one by one
    for (int i = 0; i<traceVector.size(); i++){
      String progress = counter +"/" +size;
      System.out.print(progress);
      counter++;
      Trace currentTrace = (Trace)traceVector.get(i);
      for (int j = 0; j<currentTrace.hops.size();j++){
        Hop currentHop = (Hop)currentTrace.hops.get(j);
        Host currentHost = currentHop.host;
        if (!hosts.contains(currentHost)){
            hosts.add(currentHost);
        }
      }
      for (int q = 0; q<progress.length();q++){
        System.out.print("\b");
      }
    }
    System.out.println("DONE         ");
  }
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class Trace implements Serializable{
  //instance variables
  Host from;
  Host to;
  Date timestamp1; //the timestamps are in milliseconds from epochtime
  Date timestamp2; //so remember to multiply by 1000!! (also add L for Long)
  Vector hops;   //First and last hop also stored here (#hops = hops.size()-1!

  //this constructor parses a trace from a string
  public Trace(String s, Vector h){
    hops = new Vector();
    StringTokenizer sttk = new StringTokenizer(s);
    int tokens = sttk.countTokens();
    String fromstr = sttk.nextToken();
    String tostr = sttk.nextToken();
    if( (!(NodeLink.testboxes.containsName(fromstr))) |
(!(NodeLink.testboxes.containsName(tostr))) ){
      System.out.println("Unknown testbox found: " +fromstr +" and/or " +tostr);
      System.out.println("Update testboxes.txt ...goodbye!");
      System.exit(1);
    }
    from = NodeLink.testboxes.getByName(fromstr);
    Hop first = new Hop(from.ip, h);   //from box is the first hop in the hop vector:
    hops.add(first);
    to = NodeLink.testboxes.getByName(tostr);
    timestamp1 = new Date (Long.parseLong(sttk.nextToken()+"000"));
    timestamp2 = new Date (Long.parseLong(sttk.nextToken()+"000"));
    //parse remaining tokens (hops)
    for (int i=0; i<(tokens-4); i++){
      IP currentip = new IP(sttk.nextToken());
      Hop hop = new Hop(currentip, h);
      hops.add(hop);
    }
  }

  public String toString(){
    return '\n' + "Trace from "+from.toString()+" to "+to.toString()
           + " in " +(hops.size()-1)+" hops." +'\n' + "valid from: "
           + timestamp1.toString() + '\n' +"      thru: " +timestamp2.toString();
  }

  public boolean containsLoop(){
    Set testhops = new HashSet();
    boolean returnval = false;
    for (int i=0; i<hops.size(); i++){
      if (!testhops.add((Hop)hops.get(i))){
        returnval = true;
      }
    }
    return returnval;
  }

  public boolean containsUnknown(){
    boolean returnval = false;
    for (int i=0; i<hops.size(); i++){
      if (((Hop)hops.get(i)).isUnknown()){
        returnval = true;
```

```
      }
    }
    return returnval;
  }

  public boolean lastIsNotTestBox(){
    return (!(((Hop)hops.lastElement()).isTestbox()));
  }
}
```

```
/** A TestBoxes object stores the known testboxes in two different hashmaps, for easy
retrieval
  */

package nodelink;

import java.io.*;
import java.util.*;

public class TestBoxes implements Serializable{
  public static HashMap testBoxesByName;
  public static HashMap testBoxesByIP;

  //this constructor creates a TestBoxes object from an input file
  public TestBoxes (File f){
    testBoxesByName = new HashMap();
    testBoxesByIP = new HashMap();
    System.out.print("Status: ");
    try {
      BufferedReader in = new BufferedReader(new FileReader(f));
      String line;
      int testBoxCount = 0;
      int lineCount = 0;
      String status ="lines parsed: " +lineCount +" : testboxes found: " +testBoxCount;
      System.out.print(status);
      while((line = in.readLine()) != null) {
        lineCount++;
        if (!line.startsWith("#")){
          testBoxCount++;
          StringTokenizer sttk = new StringTokenizer(line);
          int tokens = sttk.countTokens();
          String name = sttk.nextToken();
          IP ip = new IP(sttk.nextToken());
          float latitude = Float.parseFloat(sttk.nextToken()+"F");
          float longitude = Float.parseFloat(sttk.nextToken()+"F");
          String description = sttk.nextToken();
          //add rest of line to description
          for (int i =0; i < tokens - 5;i++){
            description = description + " " + sttk.nextToken();
          }
          Host tb = new Host(name, ip, latitude, longitude, description);
          testBoxesByName.put(tb.name, tb);
          testBoxesByIP.put(tb.ip, tb);
        }
        for (int q = 0; q<status.length();q++){
          System.out.print("\b");
        }
        status = "lines parsed: " +lineCount +" : testboxes found: " +testBoxCount;
        System.out.print(status);
      }
      System.out.println("");
    }
    catch (IOException e){
      System.out.print("ERROR: " +e);
      System.exit(1);
    }
  }

  public boolean containsIP(IP ip){
    return testBoxesByIP.containsKey(ip);
  }

  public boolean containsName(String name){
    return testBoxesByName.containsKey(name);
  }

  public Host getByName(String name){
    return (Host)testBoxesByName.get(name);
  }

  public Host getByIP(IP ip){
    return (Host)testBoxesByIP.get(ip);
  }
```

```
  public String toString(){
    return "\nBy Name: \n" + testBoxesByName.toString()
            +"\nBy IP: \n" + testBoxesByIP.toString();
  }
}
```

Appendix C. Updated parts of the code for NodeLink_0.8

```
package nodelink;

import java.io.*;
import java.util.*;

public class NodeLink{
  //Default file locations:
  static String dataDir = ".\\data";  //".\\data\\test for testing
  static String testboxFile = dataDir+ "\\testboxes.txt";
  static String traceDir = dataDir+ "\\tracefiles\\";
  static String whoisDir = dataDir+ "\\whois\\";
  static String asHashFile = dataDir+ "\\hashtable.dat";
  static String otterDir = dataDir+ "\\odf\\";
  //variables
  static TestBoxes testboxes;
  static Traces traces;
  static ASHash ASHashtable;

  public static void main(String[] args){
    //first read in testboxes from file
    System.out.println("Will first read the testboxes from: " +testboxFile);
    testboxes = new TestBoxes(new File(testboxFile));
    //next collect the traces
    System.out.println("\nEnter filename of inputfile(will look in " +traceDir +"): ");
    String filename = traceDir + DialogTools.getInput();
    traces = new Traces(filename);
    traces.stats();
    System.out.print("\nFilter out traces that contain loops? (y/other)");
    boolean loop = DialogTools.getAnswer();
    System.out.print("Filter out traces that contain unknown hops? (y/other)");
    boolean unknown = DialogTools.getAnswer();
    System.out.print("Filter out traces where last hop is no testbox? (y/other)");
    boolean testbox = DialogTools.getAnswer();
    if (loop | unknown | testbox){
      traces.filter(loop, unknown, testbox);
      traces.stats();
    }
    //Now create a new Graph of the traceVector
    System.out.println("\nWill now create graph from trace data");
    Graph ipGraph = new Graph(traces, false);
    System.out.println("\nWill now export Graph to ODF file");
    System.out.print("Enter filename (will place in "+otterDir+"):");
    String file = DialogTools.getInput();
    ipGraph.exportToOdf(otterDir+file);
    //next create an ASHash object, either load or create new
    System.out.println("[r]ead hashtable from file or create new hashtable [any other key]?");
    String answer = DialogTools.getInput();
    if(answer.startsWith("r")){
      //this constructor loads the hashtable from the ASHashFile
      ASHashtable = new ASHash();
    }
    else {
      ASHashtable = new ASHash(whoisDir);
    }
    //Now use this hashtable to map the hosts onto their asn's
    System.out.println("\nWill now try to map al the unique hosts onto an asn:");
    int mapped =0;
    int failed =0;
    int total = 0;
    int count =1;
    int size = traces.hosts.size();
    String progress = "Host " +count +" of " +size +": Mapped :" +mapped +" Failed: " +failed;
    for (int i=0; i<size;i++){
      total++;
      Host currentHost = (Host)traces.hosts.get(i);
      int result = ASHashtable.search(currentHost.ip);
      if (!(result == 0)){
        currentHost.setAsn(result);
        mapped++;
      }
      else {
        failed++;
      }
      progress = "Host " +count +" of " +size +" mapped :" +mapped +" failed: " +failed;
      for (int q = 0; q<progress.length();q++){
        System.out.print("\b");
      }
      System.out.print(progress);
```

```
      count ++;
    }
    System.out.println("");
    System.out.println("Writing hosts + mappings to : hosts.txt");
    File f = new File ("hosts.txt");
    try{
      PrintWriter p = new PrintWriter(new FileWriter(f));
      for (int i = 0; i< traces.hosts.size();i++){
        p.println(((Host)traces.hosts.elementAt(i)).name +" : "
+((Host)traces.hosts.elementAt(i)).asn);
      }
      p.close();
    }
    catch (Exception e){
      System.out.println(e);
    }
//     for (int i=0; i<traces.hosts.size();i++){
//       System.out.println(((Host)traces.hosts.get(i)).ip +" : "
+((Host)traces.hosts.get(i)).asn);
//     }
    //Next create a new reduced Graph!
    System.out.println("\nWill now create a reduced graph from trace data");
    //false: bewaar de niet gemapte hosts ook in de odf file
    //true: gooi de niet gemapte hosts uit de odf file
    Graph asGraph = new Graph(traces, false);
    System.out.println("\nWill now export Graph to ODF file");
    System.out.print("Enter filename (will place in "+otterDir+"):");
    file = DialogTools.getInput();
    ipGraph.exportToOdf(otterDir+file);
  }
}
```

```
package nodelink;

import java.io.*;
import java.util.*;

public class Node implements Serializable {
  String name;
  float latitude;
  float longitude;
  boolean reduced;
  //for providing a location on the grid in Otter, currently not used
  //int x;
  //int y;

  public Node(Host host){
    if (host.isMapped()){
      this.name = String.valueOf(host.asn);
      this.reduced = true;
      this.latitude =0;
      this.longitude =0;
    }
    else {
      this.name = host.name;
      this.reduced = false;
      this.latitude = host.latitude;
      this.longitude = host.longitude;
    }

  }

  public boolean hasGeo(){
    return (!(latitude ==0 && longitude ==0));
  }

  public String toOtterGeoString(){
    return String.valueOf(latitude) +" " +String.valueOf(longitude)+" " +name;
  }

  public String toOtterString(){
    return name;
  }

  public Node(Hop hop){
    this.name = hop.host.name;
    this.latitude = hop.host.latitude;
    this.longitude = hop.host.longitude;
  }

  public String toString(){
    return name;
  }
```

```
  public boolean equals(Object obj) {
    //first check if it is a Node
    if (!(obj instanceof Node)){
      return false;
    }
    else {
      //two Nodes are equal if their names are the same
      return ((Node)obj).name.equals(name);
    }
  }

   public int hashCode(){
    return name.hashCode();
  }

}
```

```
/** this class defines a Graph object. A Graph consists of a Vector of nodes
  * and a Vector of links. Among its methods are exportToODF and reduce
  */

package nodelink;

import java.io.*;
import java.util.*;

public class Graph {
  //instance variables
  static Vector nodes;
  static Vector links;

  //constructor
  public Graph(){
    nodes = new Vector();
    links = new Vector();
  }
  //constructor
  public Graph(Vector nodes, Vector links){
    this.nodes = nodes;
    this.links = links;
  }

  //constructor based on a traces object
  public Graph (Traces traces, boolean reduced){
    nodes = new Vector();
    links = new Vector();
    //first collect all the unique nodes, which are in the traces.hosts Vector.
    System.out.print("collecting nodes: ");
    int nodecounter = 1;
    int size = traces.hosts.size();
    for (int i=0;i<traces.hosts.size();i++){
      String progress = nodecounter +"/" +size;
      System.out.print(progress);
      Node node = new Node((Host)traces.hosts.get(i));
      if (reduced){
        if (node.reduced){
          if(!(nodes.contains(node))){
            nodes.add(node);
            nodecounter++;
          }
        }
      }
      else {
        if(!(nodes.contains(node))){
          nodes.add(node);
          nodecounter++;
        }
      }

      for (int q = 0; q<progress.length();q++){
        System.out.print("\b");
      }
    }
    //next get the links
    System.out.print("\ncollecting links from trace: ");
    size = traces.traceVector.size();
    int counter = 1;
    for (int i = 0; i<traces.traceVector.size(); i++) {
      String progress = counter +"/" +size;
      System.out.print(progress);
      //Take every trace in the traceVector one by one
      Trace currentTrace = (Trace)traces.traceVector.get(i);
      for (int j = 0 ; j<(currentTrace.hops.size()-1);j++){
        Hop hop1 = (Hop)currentTrace.hops.get(j);
```

```
      Hop hop2 = (Hop)currentTrace.hops.get(j+1);
      Node node1 = new Node((Host)traces.hosts.get(traces.hosts.indexOf(hop1.host)));
      Node node2 = new Node((Host)traces.hosts.get(traces.hosts.indexOf(hop2.host)));
      addLink(node1, node2, reduced);
    } //end of this hopvector
    counter++;
    for (int q = 0; q<progress.length();q++){
      System.out.print("\b");
    }
  }  //end of this traceVector
  System.out.println("\nDONE!         ");
  System.out.println("Nodes in graph             : "+nodes.size());
  System.out.println("Links in graph             : "+links.size());
}

public boolean addLink(Node node1, Node node2, boolean reduced){
  if (reduced){
    if ((!(node1.reduced)) | (!(node2.reduced)) ){
      return false;
    }
  }
  if (node1.equals(node2)){
    //System.out.println("Link A-A type");
    return false;
  }
  else {
    Link link = new Link(node1,node2);
    if (links.contains(link)){
      //System.out.println("Link already in links!");
      return false;
    }
    else {
      Link reverse = new Link(node2, node1);
      if (links.contains(reverse)){
        //System.out.println("Reverse link already in links. Set bidirectional true!");
        ((Link)links.get(links.indexOf(reverse))).setBidirectional(true);
        return false;
      }
      else {
        links.add(link);
        return true;
      }
    }
  }
}

public void exportToOdf(String filename){
  //first create a new "otterlinks" Vector
  Vector otterlinks = new Vector();
  //then examine each link
  for (int i = 0; i<links.size(); i++){
    //get the positions of the nodes of the link in the nodes vector
    Link link = (Link)links.get(i);
    String pos1 = Integer.toString(nodes.indexOf(link.node1));
    String pos2 = Integer.toString(nodes.indexOf(link.node2));
    //create a string with the two positions
    //Todo, something with bidirectional link!
    String otterlink = pos1 +" "+pos2;
    otterlinks.add(otterlink);
  }
  //next write the odf file
  System.out.println("Writing ODF file:");
  try {
    File f = new File (filename);
    PrintWriter p = new PrintWriter(new FileWriter(f));
    p.println("t " +nodes.size());
    p.println("T " +otterlinks.size());
    //first the nodes
    System.out.print("Writing nodes: ");
    for (int i = 0; i<nodes.size();i++){
      String progress = (i+1) +"/" +nodes.size();
      System.out.print(progress);
      Node currentNode = (Node)nodes.get(i);
      if (currentNode.hasGeo()){
        p.println("N " +i +" " +currentNode.toOtterGeoString());
      }
      else {
        p.println("? " +i +" " +currentNode.toOtterString());
      }
      for (int q = 0; q<progress.length();q++){
        System.out.print("\b");
      }
    }
    System.out.println(nodes.size() +" written!       ");
```

```
      //then the links
      System.out.print("Writing links: ");
      for (int i = 0; i<otterlinks.size();i++){
        String progress1 = (i+1) +"/" +otterlinks.size();
        System.out.print(progress1);
        p.println("l " +i +" " +otterlinks.get(i).toString());
        for (int q = 0; q<progress1.length();q++){
          System.out.print("\b");
        }
      }
      System.out.println(links.size() +" written!       ");
      p.close();
      System.out.println("DONE!");
    }
    catch (IOException e){
      System.out.println("[ERROR]");
      System.out.println(e);
      System.exit(1);
    }
  }
}
```

```
/** this class defines the IPRange objects. An IPRange instance contains
  * an IP and a prefixlength  */
package nodelink;

import java.io.*;
import java.util.*;

public class IPRange implements Serializable {
  //instance variables
  public IP ip;
  public int prefix;

  //constructor
  public IPRange(){
  }

  //constructor
  public IPRange(IP ip, int prefix){
    this.ip = ip;
    this.prefix = prefix;
  }

  //constructor
  public IPRange(String str){
    this.ip = new IP(str.substring(0, str.indexOf("/")));
    this.prefix = Integer.parseInt(str.substring(str.indexOf("/")+1));
  }

  public String toString() {
    return ip.toString() + "/" + prefix;
  }

//These next two methods overide the equals and hashCode methods
//in Object. This is needed to compare two IPRange objects, not based
//on the actual objects, but on the values of their variables. Two IPRanges
//that have the same ip and prefix values are said to be equal!
//This is needed in order to search in the Hashtable
//(If you overide equals then you MUST overide hashCode as well!
// Look at the java tutorial:
// http://java.sun.com/docs/books/tutorial/java/javaOO/objectclass.html
// for more on this!!)

  public boolean equals(Object obj) {
    //first check of obj really is an IPRange instance
    if (!(obj instanceof IPRange)){
      return false;
    }
    boolean test_ip = ((IPRange)obj).ip.equals(ip);   // note the cast to IPRange!!
    boolean test_prefix = ((IPRange)obj).prefix == prefix;
    boolean returnval = test_ip & test_prefix;
    return returnval;
  }

  public int hashCode(){
    int returnval = (int)(ip.toDec()*prefix);
    return returnval;
  }

  //this method checks if an IP address is contained within this IPRange
  public boolean containsIP(IP inIP){
    long decIP = inIP.toDec();
    long shift = (long)(Math.pow(2,32-prefix));
```

```
    long decmask = (long)((Math.pow(2,prefix)-1)*shift);
    IP compare = new IP ((long)decIP & decmask);
    return compare.equals(this.ip);
  }

  //this method checks if an IPRange is contained within this IPRange
  public boolean containsRange(IPRange inIPRange){
    if (inIPRange.prefix < this.prefix){
      return false;
    }
    else {
      return this.containsIP(inIPRange.ip);
    }
  }
}
```

```
/**this class defines a ASHash object, that contains a hashtable of
 * IPRange objects mapped to Integer objects(!) (the ASN).
 * Its most important method is the search method, which takes
 * an IPString object and returns the corresponding ASN.
 */

package nodelink;

import java.io.*;
import java.util.*;

public class ASHash implements Serializable{
  //class variables
  public static Hashtable ASHashtable;
  public static Vector Inspect;
  public static Vector Inconsist;

  //constructor
  public ASHash(){
    try{
      ASHashtable = readHash();
    }
    catch (Exception e){
      System.out.println(e);
    }
  }

  //this constructor creates a new ASHash object by parsing the whois results
  public ASHash(String whoisDir){
    //an array to store the files in the whois dir
    File[] files;
    System.out.println ("\nWill now create a new ASHashtable:");
    System.out.println ("Whois data directory (input)  = " +whoisDir);
    //instantiate the hashtable and the vector
    this.ASHashtable = new Hashtable();
    Inspect = new Vector();
    Inconsist = new Vector();
    //get the list of files in the whoisDir
    try {
      files = getFileList(whoisDir);
    }
    catch (IOException e) {
      System.out.println("error reading file list from directory. Goodbye!");
      files = null;
      System.exit(1);
    }
    try {
      processFiles(files);
    }
    catch (IOException e){
      System.out.println("error processing files. Goodbye!");
      files = null;
      System.exit(1);
    }
  }
/*This method returns an array of Files in a given directory.
 *Remarks: Also directories are returned so make sure no subdirs are present
 */
  public static File[] getFileList(String dirname) throws IOException {
    File dir = new File(dirname);
    if (!dir.isDirectory()) {
      System.out.println(dirname+ " is not a directory!");
      File[] arrayOfFiles = {};                  //create empty array
      return arrayOfFiles;
    }
    else {
      File[] arrayOfFiles = dir.listFiles();     //create array with files
       return arrayOfFiles;
```

```
    }
  }

/* This method processes a file, line by line. This method calls
 * other methods: discardLines looks if a line doesn't contain an
 * IP address, these lines can be discarded.
 * This method looks for the IPRange
 * and tries to store it in the hashtable. If this doesn't
 * work it wil be stored in the manualInspect Vector
 */

  public static void processFiles(File[] files) throws IOException {
    //some statistics
    int lines =0;                          //total lines in file
    int strange =0;                        //no good parser yet, to inspect Vector
    int discarded =0;                      //discarded lines
    int inconsistencies =0;                //IPRange already mapped to other AS
    int doubles =0;                        //IPRange already mapped to same AS
    int parsed = 0;                        //parsed lines --> added to hashtable
    System.out.print("Working on file: ");
    int count =1;
    int size = files.length;
    for (int i=0; i<files.length; i++){
      String progress = count +"/" +size;
      count ++;
      System.out.print(progress);
      File f = files[i];
      int pathLength = f.getParent().length();
      String fileName = f.toString().substring(pathLength+1);
      String asnstr = fileName.substring(2,fileName.indexOf("."));
      Integer currentAS = new Integer(asnstr);
      BufferedReader in = new BufferedReader(new FileReader(f));
      String line;
      int lineInFile =0;
      while((line = in.readLine()) != null) {
        lines ++;
        lineInFile ++;
        //first check if there are three dots in this line (possible IPaddress)
        int dots =0;
        for(int j=0;j<line.length();j++){
          if (line.charAt(j) =='.'){
            dots +=1;
          }
        }
        if (!(dots>2)){
          discarded++;
          continue;
        }
        //if the line "makes it" to this point --> use some more filters
        if (line.startsWith("changed:")){
          discarded++;
          continue;
        }
        if (line.startsWith("*ch:")){
          discarded++;
          continue;
        }
        if (line.startsWith("*ny:")){
          discarded++;
          continue;
        }
        if (line.startsWith("notify:")){
          discarded++;
          continue;
        }
        if (line.startsWith("descr:")){
          discarded++;
          continue;
        }
        if (line.startsWith("*de:")){
          discarded++;
          continue;
        }
        //finally try to parse this line
        //note: 3 times the same piece of code, needs to be cleaned up a bit
        if (line.startsWith("*rt:")) {
          IPRange range = new IPRange(line.substring(4).trim());
          if(!ASHashtable.containsKey(range)){
            ASHashtable.put(range,currentAS);
            parsed++;
          }
          else if((ASHashtable.get(range)).equals(currentAS)){
            doubles +=1;                   //same mapping already in hashtable
          }
```

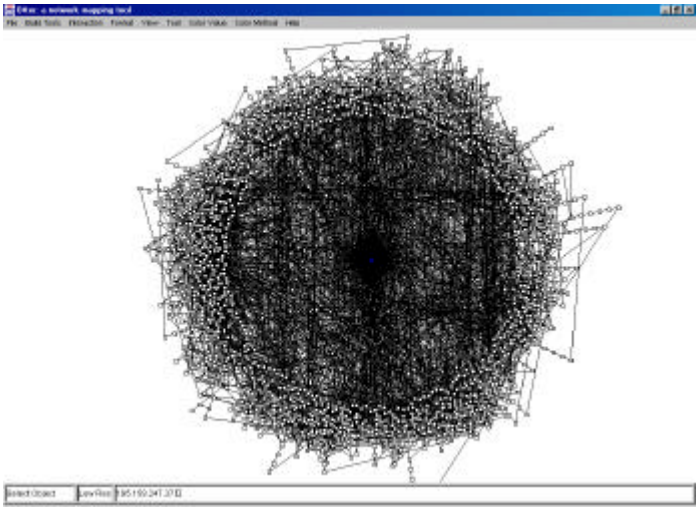
```
          else {
            //key already in hashtable, but different mapping
            String one = "Found: "+ASHashtable.get(range).toString()+" : " +range.toString();
            String two = "Using: "+asnstr +" : " +range.toString();
            Inconsist.addElement(one);
            Inconsist.addElement(two);
            //Intermediate solution: remove the existing mapping and store the last one
found...
            ASHashtable.remove(range);
            ASHashtable.put(range,currentAS);
            inconsistencies +=1;
          }
          continue;
        }
        if (line.startsWith("route:")) {
          IPRange range = new IPRange(line.substring(6).trim());
          if(!ASHashtable.containsKey(range)){
            ASHashtable.put(range,currentAS);
            parsed++;
          }
          else if((ASHashtable.get(range)).equals(currentAS)){
            doubles +=1;                  //same mapping already in hashtable
          }
          else {
            //key already in hashtable, but different mapping
            String one = "Found: "+ASHashtable.get(range).toString()+" : " +range.toString();
            String two = "Using: "+asnstr +" : " +range.toString();
            Inconsist.addElement(one);
            Inconsist.addElement(two);
            //Intermediate solution: remove the existing mapping and store the last one
found...
            ASHashtable.remove(range);
            ASHashtable.put(range,currentAS);
            inconsistencies +=1;
          }
          continue;
        }
        /* This parser causes errors, this needs some work!
        if (line.substring(15).startsWith("ias-int:")) {
          String iasint = line.substring(15).substring(9);
          String rangeString = iasint.substring(0,iasint.indexOf("AS")).trim().concat("/32");
          IPRange range = new IPRange(rangeString);
          Integer as = new Integer(iasint.substring(iasint.indexOf("AS")+2).trim());
          currentAS = as;
          if(!ASHashtable.containsKey(range)){
            ASHashtable.put(range,currentAS);
            parsed++;
          }
          else if((ASHashtable.get(range)).equals(currentAS)){
            doubles +=1;                  //same mapping already in hashtable
          }
          else {
            //key already in hashtable, but different mapping
            String one = "Found: "+ASHashtable.get(range).toString()+" : " +range.toString();
            String two = "Using: "+asnstr +" : " +range.toString();
            Inconsist.addElement(one);
            Inconsist.addElement(two);
            //Intermediate solution: remove the existing mapping and store the last one
found...
            ASHashtable.remove(range);
            ASHashtable.put(range,currentAS);
            inconsistencies +=1;
          }
          continue;
        }
        */
        //if none of the filters or parsers apply, add this line to the inspect vector!
        strange++;
        String l = fileName +" line: " +lineInFile +" : " +line;
        Inspect.add(l);
      }
      in.close();
      for (int q = 0; q<progress.length();q++){
        System.out.print("\b");
      }
    }
    System.out.println ("\nTotal lines processed : " +lines);
    System.out.println ("Total lines discarded : " +discarded);
    System.out.println ("Total IPRanges added  : " +parsed);
    System.out.println ("Lines to be inspected : " +strange);
    System.out.println("number of keys in Hashtable :" +ASHashtable.size());
    System.out.println("inconsistencies :" +inconsistencies);
    System.out.println("double entries :" +doubles);
    System.out.println("Writing inspect vector to : inspect.txt");
```

```
    File f = new File ("inspect.txt");
    PrintWriter p = new PrintWriter(new FileWriter(f));
    for (int i = 0; i< Inspect.size();i++){
      p.println(Inspect.elementAt(i));
    }
    p.close();
    System.out.println("Writing inconsistent vector to : inconsistent.txt");
    File g = new File ("inconsistent.txt");
    PrintWriter q = new PrintWriter(new FileWriter(g));
    for (int i = 0; i< Inconsist.size();i++){
      q.println(Inconsist.elementAt(i));
    }
    q.close();
    System.out.println("Will now write ASHashtable to : "+NodeLink.asHashFile);
    try {
      writeHash();
    }
    catch (Exception e){
      System.out.println(e);
    }
  }


/* Method to look for the given IP address in the Hashtable, starting at the
 * ipAddress/32 (most specific) range and ending at the ipAddress/1
 * range(less specific). This way, the most specific match will always be
 * found first, even if the hashtable is not ordered in this way
 * If no match is found, an ASN of 0 is returned.
 */

 public int search(IP inIP) {
   boolean found = false;
   long decIP = inIP.toDec();
   int asn = 0;          //if nothing is found, zero will be returned
   for (int i = 32; i>0 && (!found); i--) {
     IPRange searchRange = createSearchRange(decIP, i);
//     System.out.println("DEBUG: ASHash.search: searchRange = " +searchRange.toString());
//     System.out.println("DEBUG: ASHash.search: get : " +(String)get(searchRange));
     if(ASHashtable.containsKey(searchRange)){
       found = true;
       asn = ((Integer)ASHashtable.get(searchRange)).intValue();
//        System.out.println("DEBUG: ASHash.search: mapping found: " +asn);
       return asn;
     }
//     System.out.println("DEBUG: ASHash.search: not found: trying less specific!");
   }
//    System.out.println("DEBUG: ASHash.search: not found: returning "+asn);
   return asn;
 }

 public IPRange createSearchRange (long inDecIP, int inPrefix){
   long shift = (long)(Math.pow(2,32-inPrefix));
   long decmask = (long)((Math.pow(2,inPrefix)-1)*shift);
   long network = inDecIP & decmask;
//   System.out.println("DEBUG: ASHash.createSearchRange: network = " +network);
   IP net = new IP(network);  //create a new IPString object
//   System.out.println("DEBUG: ASHash.createSearchRange: network in dotted decimal = "
+net.toString());
   IPRange returnRange = new IPRange(net, inPrefix);
//   System.out.println("DEBUG: ASHash.createSearchRange: IPRange = "
+returnRange.toString());
   return returnRange;
 }

 public String toString(){
   return ASHashtable.toString();
 }

   //This method reads the hashtable from a file
  public static Hashtable readHash() throws IOException, ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(NodeLink.asHashFile));
    Hashtable hash = (Hashtable) in.readObject();
    in.close();
    return hash;
  }

  /* This method writes a hashtable to a file */
  public static void writeHash() throws IOException {
    ObjectOutputStream out = new ObjectOutputStream (new
FileOutputStream(NodeLink.asHashFile));
    out.writeObject(ASHashtable);
    out.close();
  }
}
```

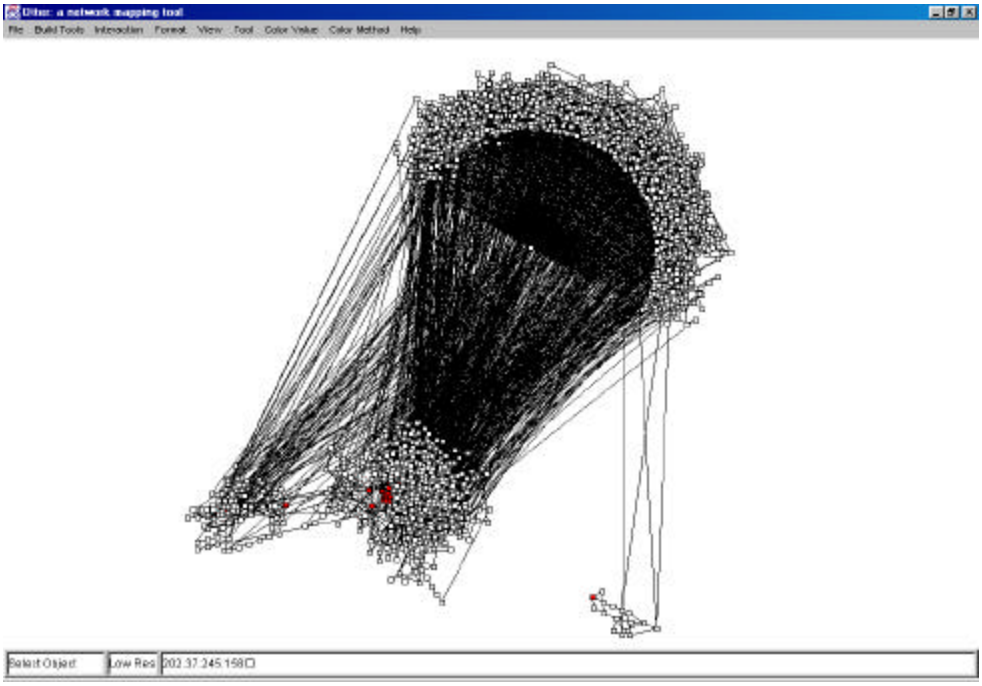
Appendix D. Results from NodeLink_0.7

20000701 dataset without filters

```
D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20000701

Reading from .\tracefiles\20000701 :line: DONE
Collecting statistics : Done

Traces in traceVector         :1185
Total Hops                    :17253
Average Hops                  :14.559494
Unique hosts (incl. Testboxes) :2302

Filter out traces that contain loops? (y/other)n
Filter out traces that contain unknown hops? (y/other)n
Filter out traces where last hop is no testbox? (y/other)n

Will now create graph from trace data
collecting nodes: 2302/2302
collecting links from trace: 1185/1185
DONE!
Nodes in graph          : 2302
Links in graph          : 3717

Will now export Graph to ODF file
Enter filename (will place in .\data\):20000701.odf
Writing ODF file:
Writing nodes: 2302 written!
Writing links: 3717 written!
DONE!

D:\umeepi\NodeLink_0.7>
```



20000701 dataset without filters (straight topological)



20000701 dataset without filters (semi-geographical)

20000701 dataset, using unknown hop (255.255.255.255) filter

D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20000701

Reading from .\tracefiles\20000701 :line: DONE
Collecting statistics : Done

Traces in traceVector :1185
Total Hops :17253
Average Hops :14.559494
Unique hosts (incl. Testboxes) :2302

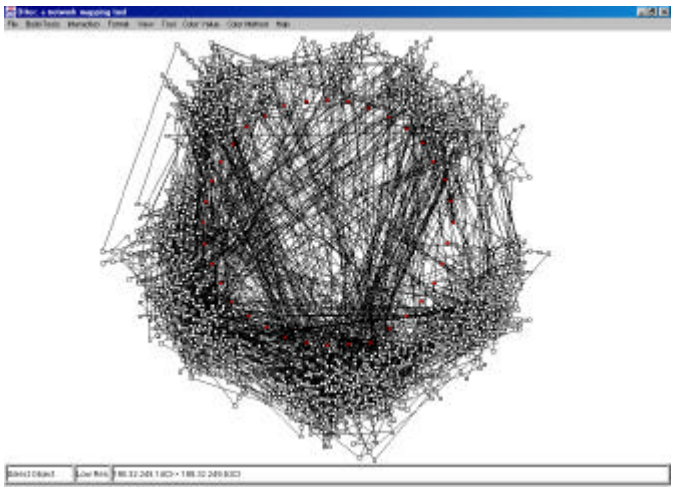
Filter out traces that contain loops? (y/other)n
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)n
Filtering: DONE

Traces with loops removed: 0
Traces with unknown hops removed: 195
Traces with last hop no testbox removed: 0
Recollecting unique hosts: DONE
Collecting statistics : Done

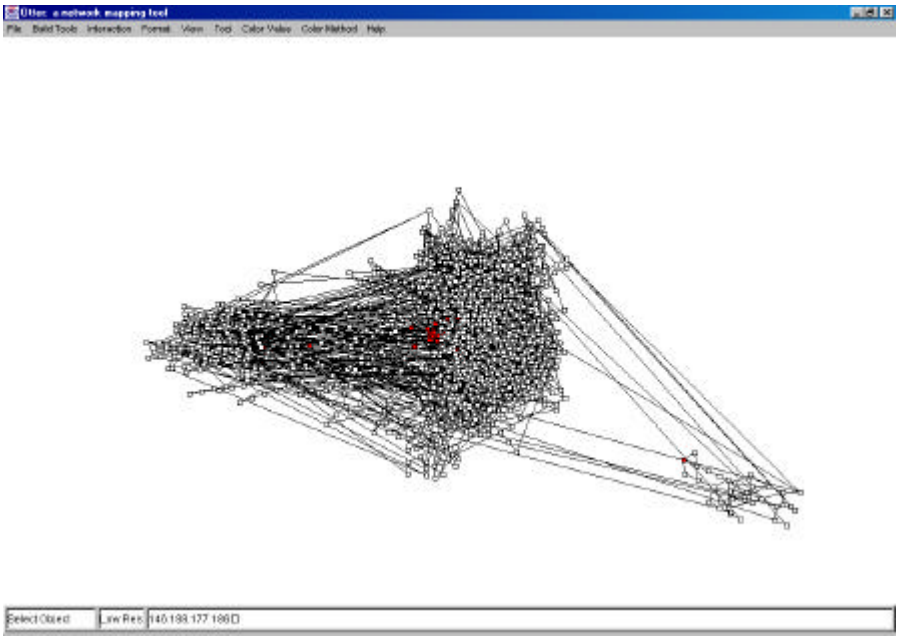
Traces in traceVector :990
Total Hops :14262
Average Hops :14.40606
Unique hosts (incl. Testboxes) :2133

Will now create graph from trace data
collecting nodes: 2133/2133
collecting links from trace: 990/990
DONE!
Nodes in graph : 2133
Links in graph : 3222

Will now export Graph to ODF file
Enter filename (will place in .\data\):20000701nyn.odf
Writing ODF file:
Writing nodes: 2133 written!
Writing li nks: 3222 written!
DONE!



20000701 dataset unknown hop filter (straight top.)



20000701 dataset unknown hop filter (semi geo)

20000701 dataset – using loop and unknown hop filter

D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20000701

Reading from .\tracefiles\20000701 :line: DONE
Collecting statistics : Done

Traces in traceVector :1185
Total Hops :17253
Average Hops :14.559494
Unique hosts (incl. Testboxes) :2302

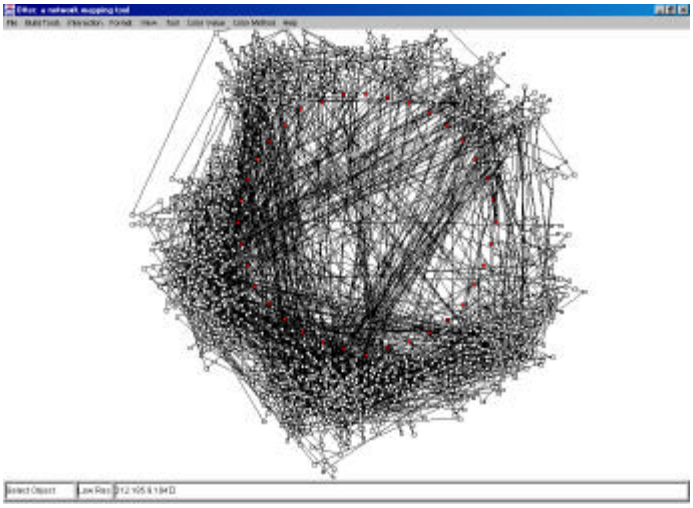
Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)n
Filtering: DONE

Traces with loops removed: 167
Traces with unknown hops removed: 88
Traces with last hop no testbox removed: 0
Recollecting unique hosts: DONE
Collecting statistics : Done

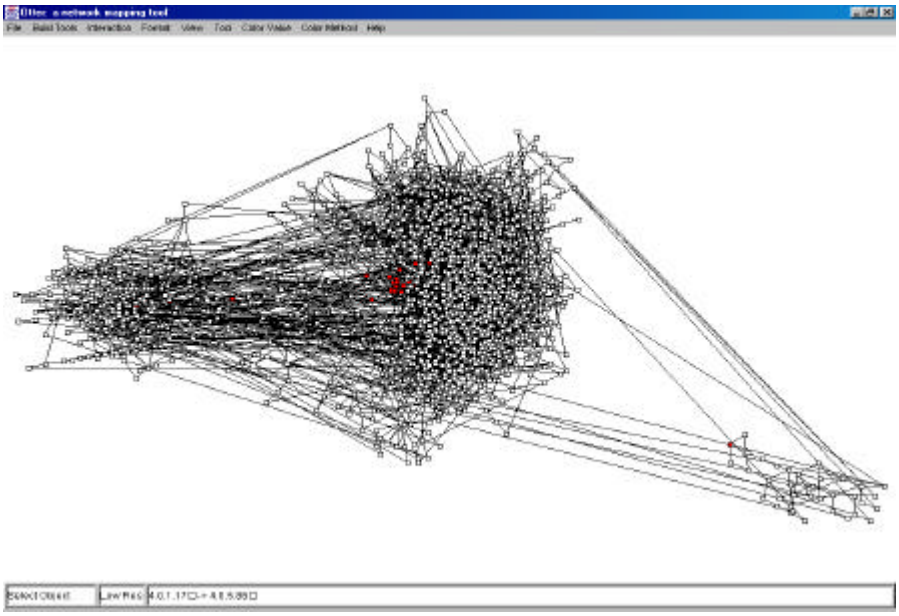
Traces in traceVector :930
Total Hops :13321
Average Hops :14.323656
Unique hosts (incl. Testboxes) :2085

Will now create graph from trace data
collecting nodes: 2085/2085
collecting links from trace: 930/930
DONE!
Nodes in graph : 2085
Links in graph : 3104

Will now export Graph to ODF file
Enter filename (will place in .\data\):20000701yyn.odf
Writing ODF file:
Writing nodes: 2085 written!
Writing links: 3104 written!
DONE!



20000701 dataset - two filters (straight topological)



20000701 dataset - two filters (semi-geo)

20000701 dataset – using all filters

```
D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20000701

Reading from .\tracefiles\20000701 :line: DONE
Collecting statistics : Done

Traces in traceVector          :1185
Total Hops                     :17253
Average Hops                   :14.559494
Unique hosts (incl. Testboxes) :2302

Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 167
Traces with unknown hops removed: 88
Traces with last hop no testbox removed: 0
Recollecting unique hosts: DONE
Collecting statistics : Done

Traces in traceVector          :930
Total Hops                     :13321
Average Hops                   :14.323656
Unique hosts (incl. Testboxes) :2085

Will now create graph from trace data
collecting nodes: 2085/2085
collecting links from trace: 930/930
DONE!
Nodes in graph             : 2085
Links in graph             : 3104

Will now export Graph to ODF file
Enter filename (will place in .\data\):20000701yyy.odf
Writing ODF file:
Writing nodes: 2085 written!
Writing links: 3104 written!
DONE!
```

Remarks: This results in the same graphs, so no extra visualisation is needed.

20000801 dataset all filters

D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20000801

Reading from .\tracefiles\20000801 :line: DONE
Collecting statistics : Done

Traces in traceVector :1181
Total Hops :17662
Average Hops :14.955123
Unique hosts (incl. Testboxes) :2336

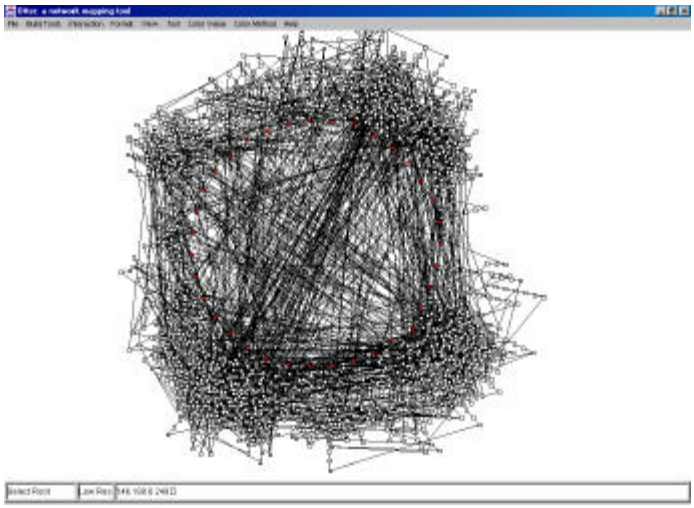
Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 151
Traces with unknown hops removed: 98
Traces with last hop no testbox removed: 0
Recollecting unique hosts: DONE
Collecting statistics : Done

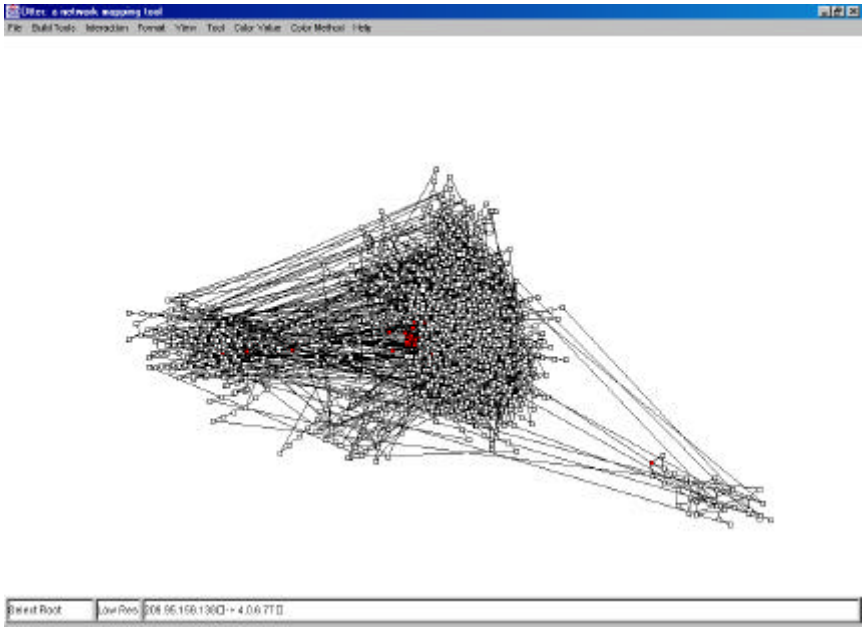
Traces in traceVector :932
Total Hops :13597
Average Hops :14.589056
Unique hosts (incl. Testboxes) :2103

Will now create graph from trace data
collecting nodes: 2103/2103
collecting links from trace: 932/932
DONE!
Nodes in graph : 2103
Links in graph : 3055

Will now export Graph to ODF file
Enter filename (will place in .\data\):20000801yyy.odf
Writing ODF file:
Writing nodes: 2103 written!
Writing links: 3055 written!
DONE!



20000801 dataset all filters (straight top.)



20000801 dataset all filters (semi-geo)

20000901 Dataset - all filters

```
D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20000901

Reading from .\tracefiles\20000901 :line: DONE
Collecting statistics : Done

Traces in traceVector         :846
Total Hops                  :13297
Average Hops                  :15.717494
Unique hosts (incl. Testboxes) :1850

Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 56
Traces with unknown hops removed: 57
Traces with last hop no testbox removed: 35
Recollecting unique hosts: DONE
Collecting statistics : Done

Traces in traceVector         :698
Total Hops                  :10381
Average Hops                  :14.872493
Unique hosts (incl. Testboxes) :1645

Will now create graph from trace data
collecting nodes: 1645/1645
collecting links from trace: 698/698
DONE!
Nodes in graph            : 1645
Links in graph            : 2371

Will now export Graph to ODF file
Enter filename (will place in .\data\):20000901yyy.odf
Writing ODF file:
Writing nodes: 1645 written!
Writing links: 2371 written!
DONE!
```

No visualisation is made here, because it doesn't really give any new insights. Graphs similar to previous ones.

20001001 dataset –all filters

D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20001001

Reading from .\tracefiles\20001001 :line: DONE
Collecting statistics : Done

Traces in traceVector :702
Total Hops :10768
Average Hops :15.339031
Unique hosts (incl. Testboxes) :1735

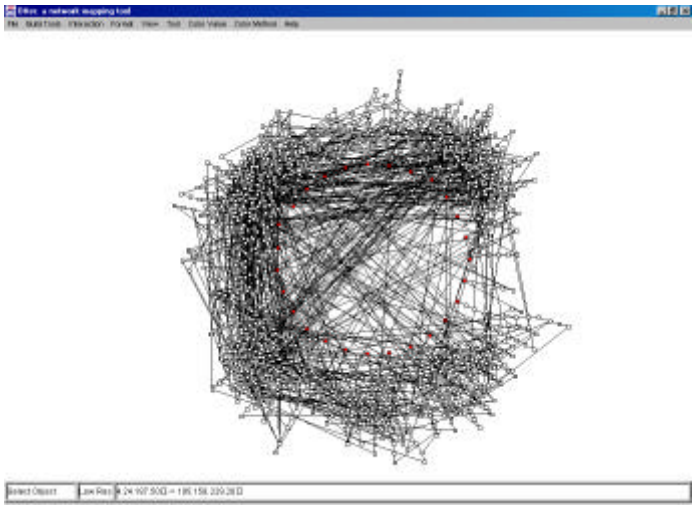
Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 88
Traces with unknown hops removed: 34
Traces with last hop no testbox removed: 9
Recollecting unique hosts: DONE
Collecting statistics : Done

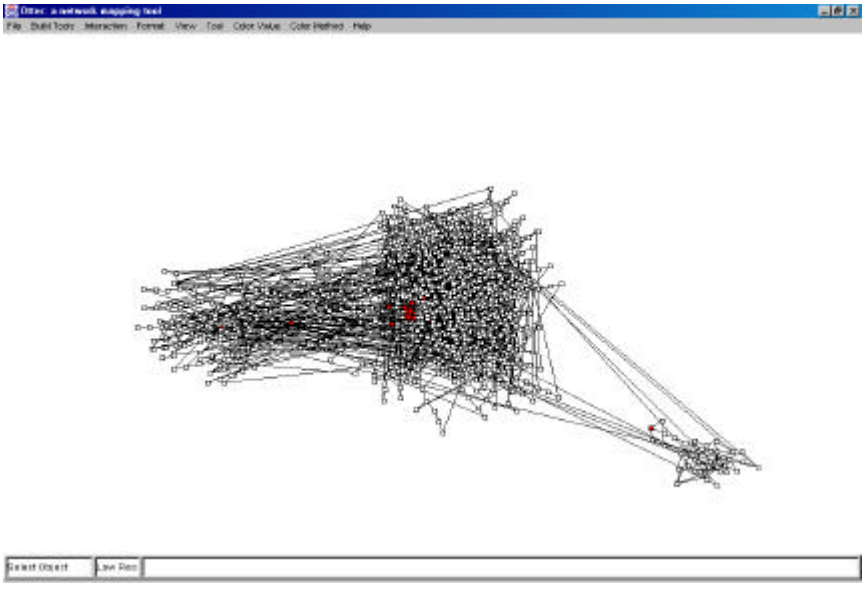
Traces in traceVector :571
Total Hops :8427
Average Hops :14.758319
Unique hosts (incl. Testboxes) :1528

Will now create graph from trace data
collecting nodes: 1528/1528
collecting links from trace: 571/571
DONE!
Nodes in graph : 1528
Links in graph : 2127

Will now export Graph to ODF file
Enter filename (will place in .\data\):20001001yyy.odf
Writing ODF file:
Writing nodes: 1528 written!
Writing links: 2127 written!
DONE!



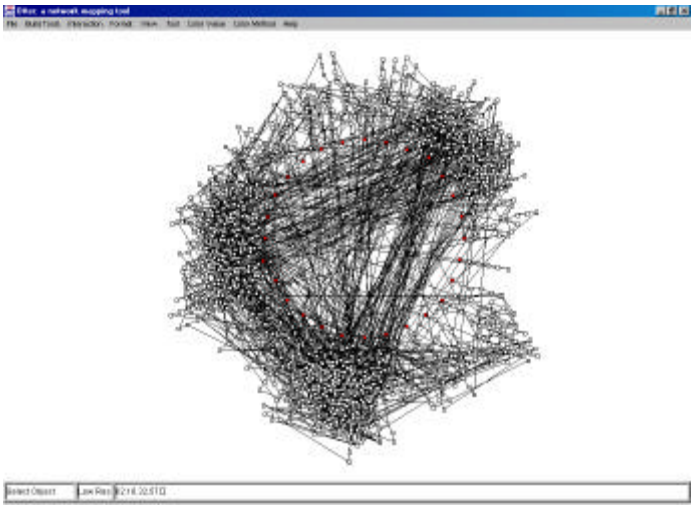
20001001 dataset - all filters (straight top)



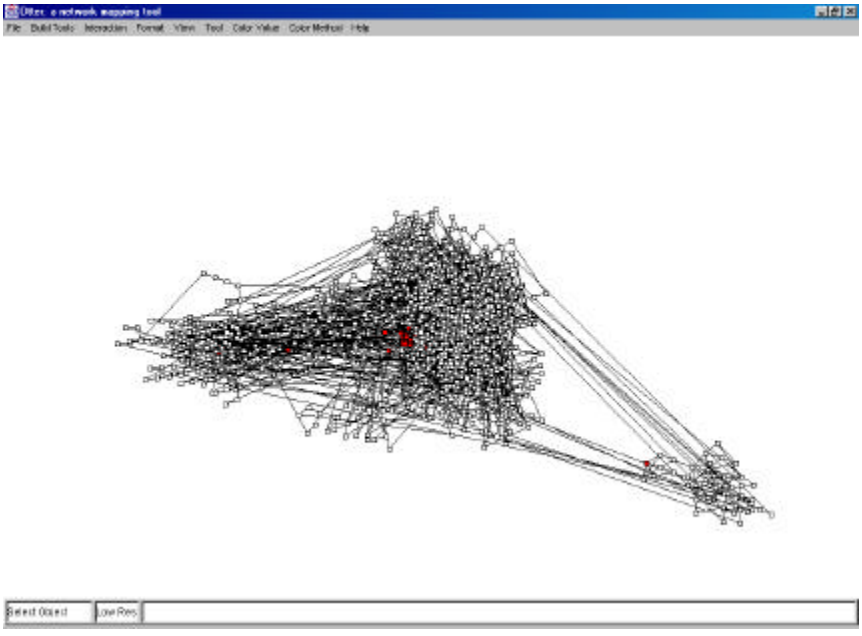
20001001 dataset - all filters (semi-geo)

20001101 dataset – all filters

```
D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20001101

Reading from .\tracefiles\20001101 :line: DONE
Collecting statistics : Done

Traces in traceVector         :757
Total Hops                    :11298
Average Hops                  :14.924703
Unique hosts (incl. Testboxes) :1825

Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 65
Traces with unknown hops removed: 41
Traces with last hop no testbox removed: 3
Recollecting unique hosts: DONE
Collecting statistics : Done

Traces in traceVector         :648
Total Hops                    :9469
Average Hops                  :14.612655
Unique hosts (incl. Testboxes) :1675

Will now create graph from trace data
collecting nodes: 1675/1675
collecting links from trace: 648/648
DONE!
Nodes in graph          : 1675
Links in graph          : 2405

Will now export Graph to ODF file
Enter filename (will place in .\data\):20001101yyy.odf
Writing ODF file:
Writing nodes: 1675 written!
Writing links: 2405 written!
DONE!
```



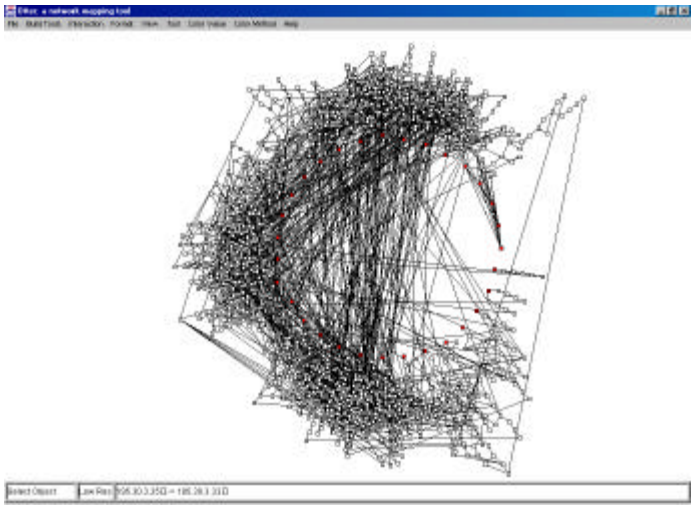
20001101 dataset - all filters (straight top)



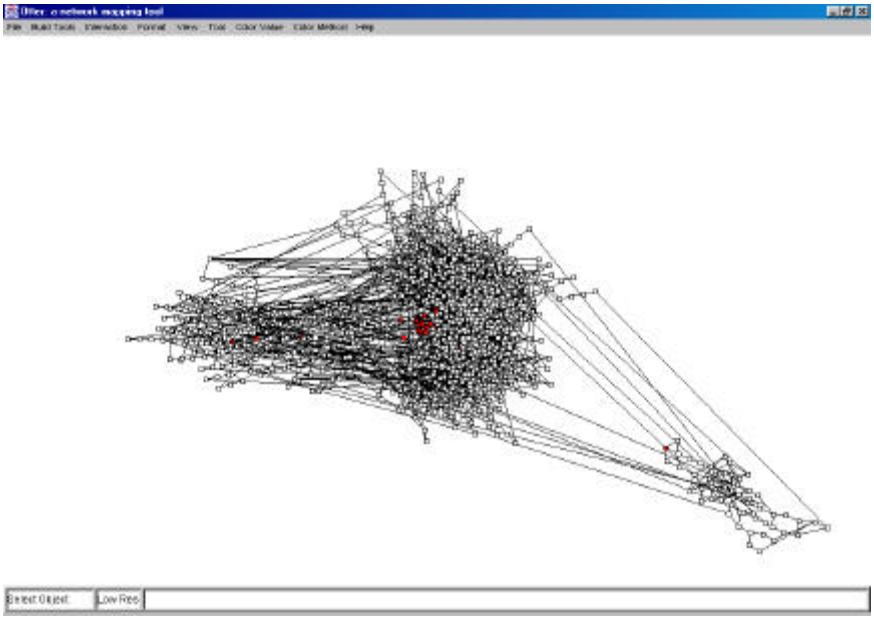
20001101 dataset - all filters (semi-geo)

20001201 dataset – all filters used

```
D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
20001201

Reading from .\tracefiles\20001201 :line: DONE
Collecting statistics : Done

Traces in traceVector         :929
Total Hops                  :13134
Average Hops                :14.137782
Unique hosts (incl. Testboxes) :1790

Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 71
Traces with unknown hops removed: 91
Traces with last hop no testbox removed: 18
Recollecting unique hosts: DONE
Collecting statistics : Done

Traces in traceVector         :749
Total Hops                  :10046
Average Hops                :13.41255
Unique hosts (incl. Testboxes) :1556

Will now create graph from trace data
collecting nodes: 1556/1556
collecting links from trace: 749/749
DONE!
Nodes in graph           : 1556
Links in graph           : 2227

Will now export Graph to ODF file
Enter filename (will place in .\data\):20001201yyy.odf
Writing ODF file:
Writing nodes: 1556 written!
Writing links: 2227 written!
DONE!
```



20001201 dataset - all filters (straight top)



20001201 dataset - all filters (semi-geo)

Combined dataset – all filters used

D:\umeepi\NodeLink_0.7>java -classpath .\class nodelink.NodeLink
Will first read the testboxes from: .\data\testboxes.txt
Status: lines parsed: 58 : testboxes found: 48

Enter filename of inputfile(will look in .\tracefiles\):
combined

Reading from .\tracefiles\combined :line: DONE
Collecting statistics : Done

Traces in traceVector :5600
Total Hops :83412
Average Hops :14.895
Unique hosts (incl. Testboxes) :4299

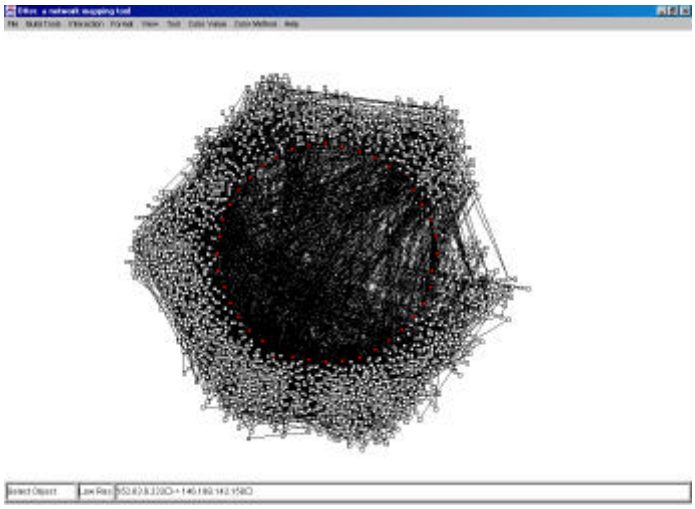
Filter out traces that contain loops? (y/other)y
Filter out traces that contain unknown hops? (y/other)y
Filter out traces where last hop is no testbox? (y/other)y
Filtering: DONE

Traces with loops removed: 598
Traces with unknown hops removed: 409
Traces with last hop no testbox removed: 65
Recollecting unique hosts: DONE
Collecting statistics : Done

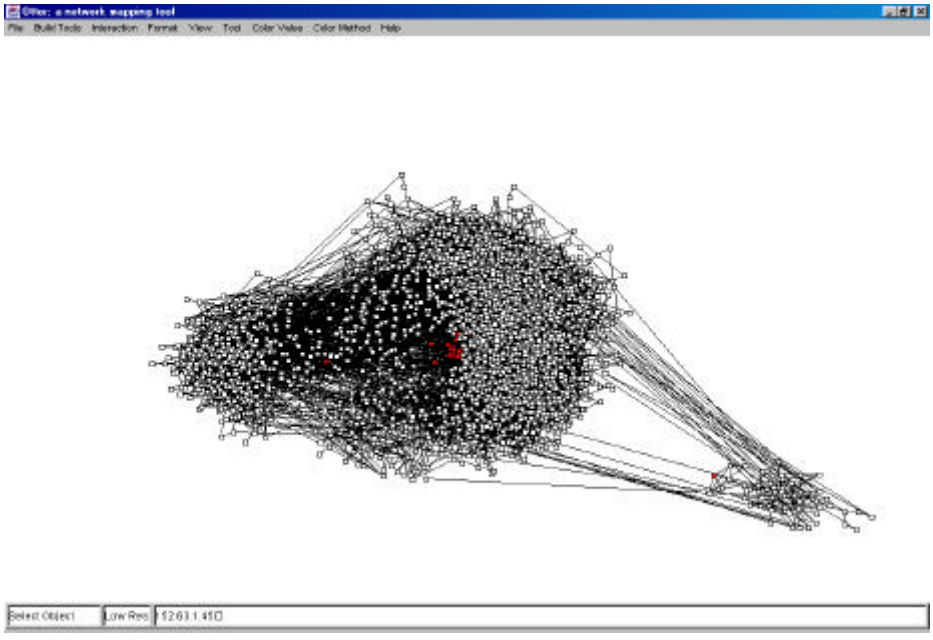
Traces in traceVector :4528
Total Hops :65241
Average Hops :14.408348
Unique hosts (incl. Testboxes) :3962

Will now create graph from trace data
collecting nodes: 3962/3962
collecting links from trace: 4528/4528
DONE!
Nodes in graph : 3962
Links in graph : 7365

Will now export Graph to ODF file
Enter filename (will place in .\data\):combinedyyy.odf
Writing ODF file:
Writing nodes: 3962 written!
Writing links: 7365 written!
DONE!



combined dataset - all filters (straight top)

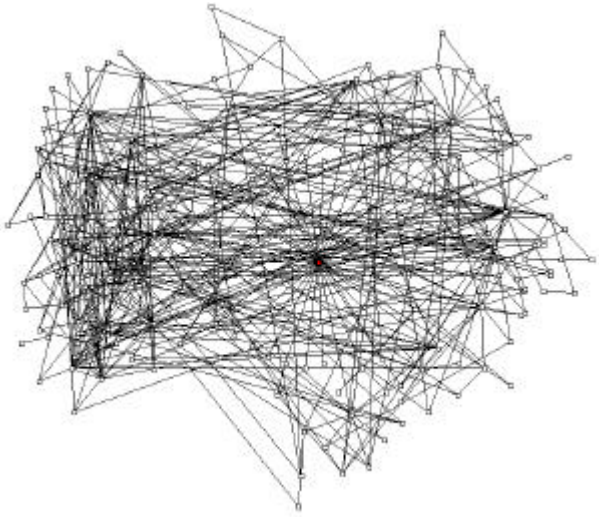


combined dataset - all filters (semi-geo)

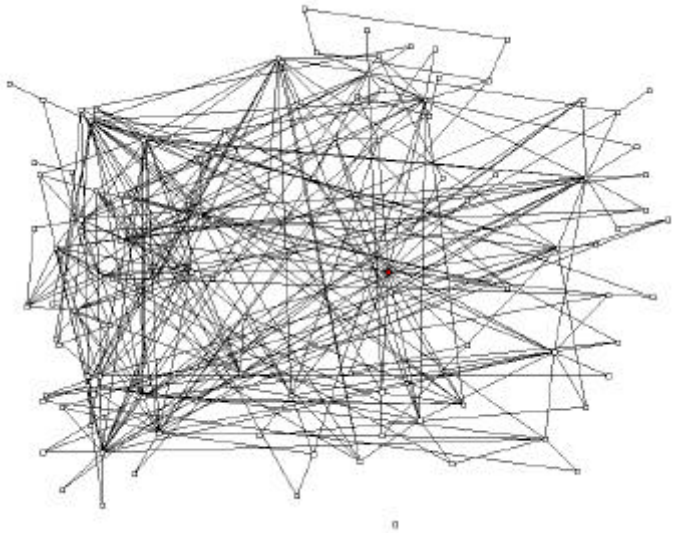
Appendix E. Results from NodeLink_0.8

20000701 dataset and hashtable creation

```
Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20000701yyy.odf
Writing ODF file:
Writing nodes: 2085 written!
Writing links: 3104 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?

Will now create a new ASHashtable:
Whois data directory (input)  = .\data\whois\
Working on file: 25850/25850
Total lines processed : 1850091
Total lines discarded : 1676734
Total IPRanges added  : 120783
Lines to be inspected : 2583
number of keys in Hashtable :120783
inconsistencies :23068
double entries :26923
Writing inspect vector to : inspect.txt
Writing inconsistent vector to : inconsistent.txt
Will now write ASHashtable to : .\data\hashtable.dat

Will now try to map al the unique hosts onto an asn:
Host 2085 of 2085 mapped :2008 failed: 77
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 190/2085
collecting links from trace: 930/930
DONE!
Nodes in graph          : 189
Links in graph          : 468

Will now export Graph to ODF file
Enter filename (will place in
.\data\odf\):20000701yyyAS+.odf
Writing ODF file:
Writing nodes: 189 written!
Writing links: 468 written!
DONE!

D:\umeepi\NodeLink_0.8>
```



AS graph of 20000701 with unmapped nodes in graph



AS graph of 20000701 with unmapped nodes removed

20000801 dataset

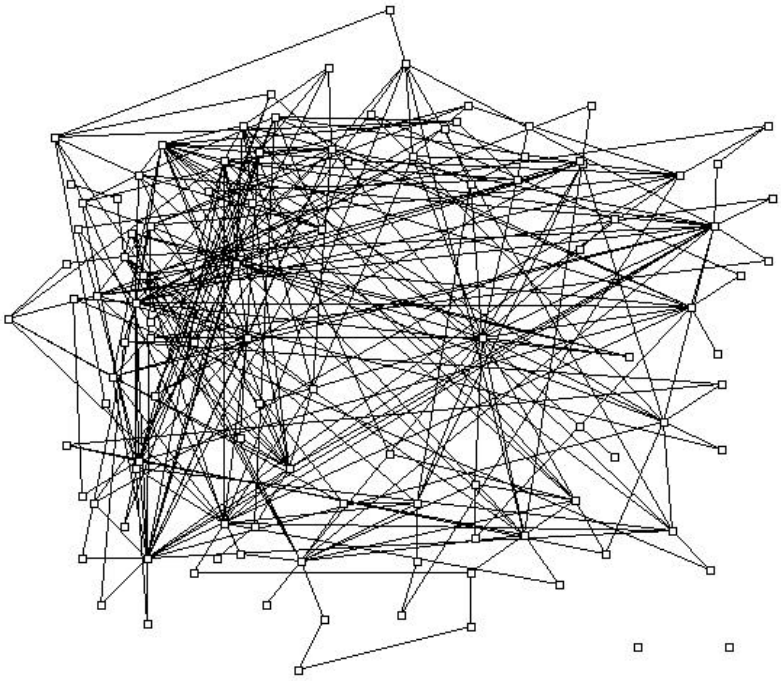
Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20000801yyyIP.odf
Writing ODF file:
Writing nodes: 2103 written!
Writing links: 3055 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?
r

Will now try to map al the unique hosts onto an asn:
Host 2103 of 2103 mapped :2028 failed: 75
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 109/2103
collecting links from trace: 932/932
DONE!
Nodes in graph : 108
Links in graph : 298

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20000801yyyAS-.odf
Writing ODF file:
Writing nodes: 108 written!
Writing links: 298 written!
DONE!

Remarks: two isolated nodes: 3228 and 8437 Central node is 1755



AS graph of 20000801 with unmapped nodes removed

20000901 dataset

Traces in traceVector :698
Total Hops :10381
Average Hops :14.872493
Unique hosts (incl. Testboxes) :1645

Will now create graph from trace data
collecting nodes: 1645/1645
collecting links from trace: 698/698
DONE!
Nodes in graph : 1645
Links in graph : 2371

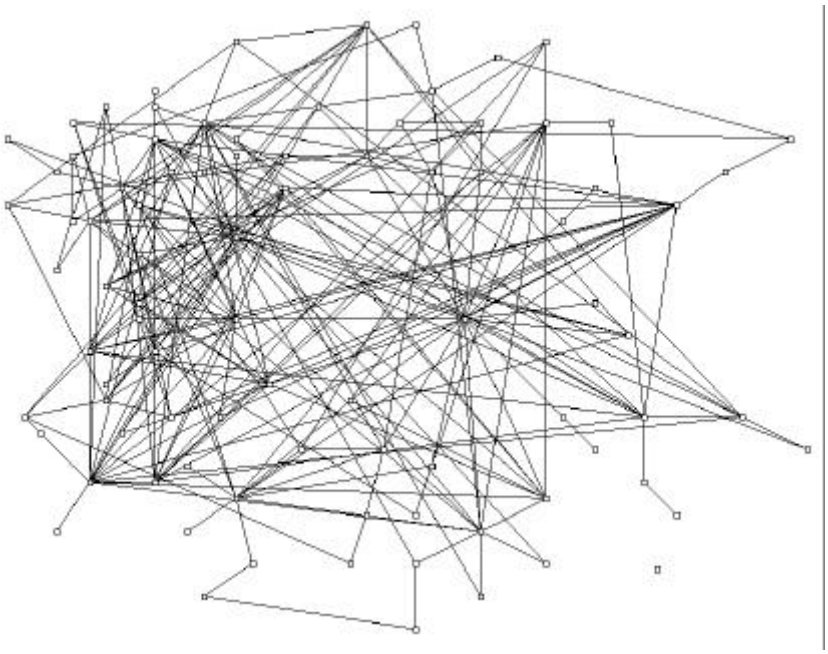
Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20000901yyyIP.odf
Writing ODF file:
Writing nodes: 1645 written!
Writing links: 2371 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?
r

Will now try to map al the unique hosts onto an asn:
Host 1645 of 1645 mapped :1571 failed: 74
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 92/1645
collecting links from trace: 698/698
DONE!
Nodes in graph : 91
Links in graph : 228

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20000901yyyAS-.odf
Writing ODF file:
Writing nodes: 91 written!
Writing links: 228 written!
DONE!

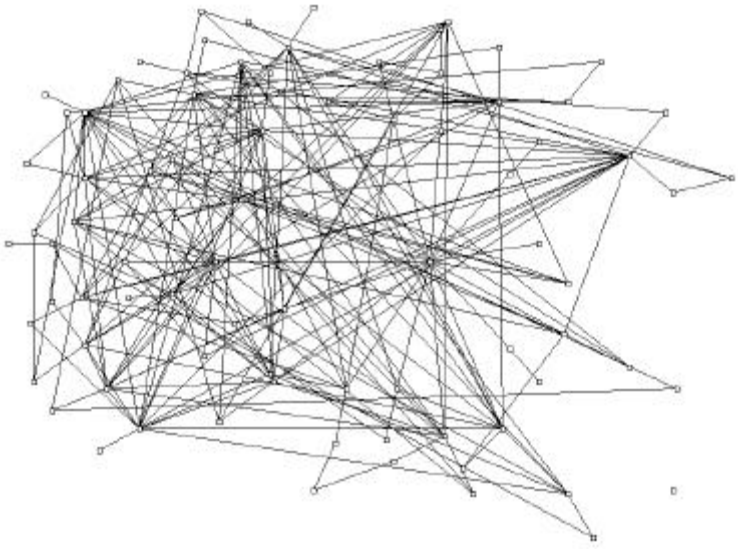
Remarks: 3228 again isolated.



AS graph of 20000901 with unmapped nodes removed

20001001 dataset

```
Traces in traceVector         :571
Total Hops                   :8427
Average Hops                  :14.758319
Unique hosts (incl. Testboxes) :1528

Will now create graph from trace data
collecting nodes: 1528/1528
collecting links from trace: 571/571
DONE!
Nodes in graph           : 1528
Links in graph           : 2127

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20001001yyyIP.odf
Writing ODF file:
Writing nodes: 1528 written!
Writing links: 2127 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?
r

Will now try to map al the unique hosts onto an asn:
Host 1528 of 1528 mapped :1437 failed: 91
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 86/1528
collecting links from trace: 571/571
DONE!
Nodes in graph           : 85
Links in graph           : 225

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20001001yyyAS-.odf
Writing ODF file:
Writing nodes: 85 written!
Writing links: 225 written!
DONE!
```



AS graph of 20001001 with unmapped nodes removed

20001101 dataset

Traces in traceVector :648
Total Hops :9469
Average Hops :14.612655
Unique hosts (incl. Testboxes) :1675

Will now create graph from trace data
collecting nodes: 1675/1675
collecting links from trace: 648/648
DONE!
Nodes in graph : 1675
Links in graph : 2405

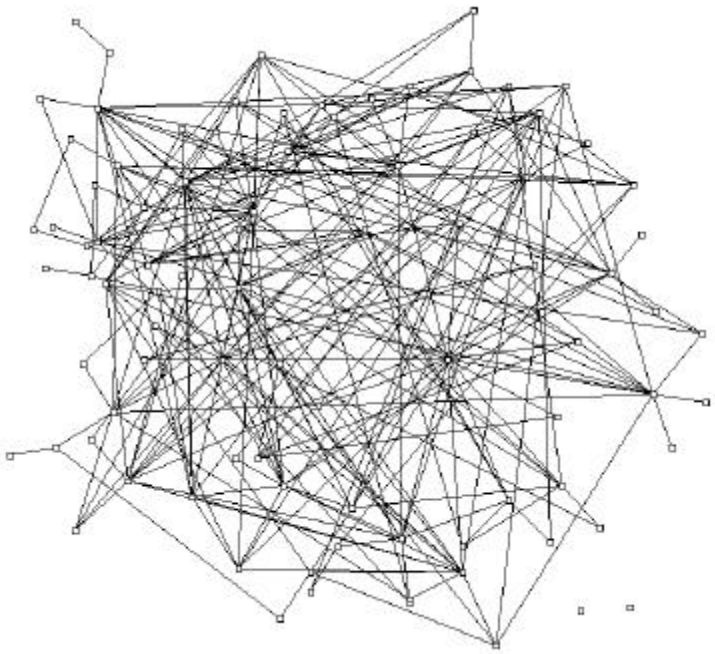
Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20001101yyy.odf
Writing ODF file:
Writing nodes: 1675 written!
Writing links: 2405 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?
r

Will now try to map al the unique hosts onto an asn:
Host 1675 of 1675 mapped :1578 failed: 97
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 90/1675
collecting links from trace: 648/648
DONE!
Nodes in graph : 89
Links in graph : 242

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20001101yyyAS-.odf
Writing ODF file:
Writing nodes: 89 written!
Writing links: 242 written!
DONE!

Remarks: Isolated: 3228 and 8284



AS graph of 20001101 with unmapped nodes removed

20001201 dataset

Traces in traceVector :749
Total Hops :10046
Average Hops :13.41255
Unique hosts (incl. Testboxes) :1556

Will now create graph from trace data
collecting nodes: 1556/1556
collecting links from trace: 749/749
DONE!
Nodes in graph : 1556
Links in graph : 2227

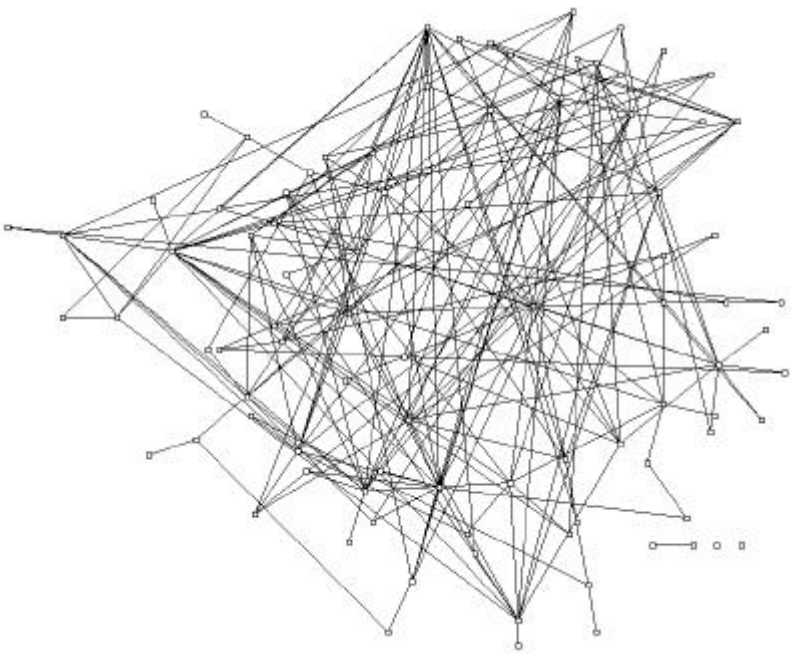
Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20001201yyyIP.odf
Writing ODF file:
Writing nodes: 1556 written!
Writing links: 2227 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?
r

Will now try to map al the unique hosts onto an asn:
Host 1556 of 1556 mapped :1462 failed: 94
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 91/1556
collecting links from trace: 749/749
DONE!
Nodes in graph : 90
Links in graph : 216

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):20001201yyyAS-.odf
Writing ODF file:
Writing nodes: 90 written!
Writing links: 216 written!
DONE!

Remarks: Two connected isolated points:
11042 and 5552
loose points: 8284 and 3228



AS graph of 20001201 with unmapped nodes removed

Combined dataset

Traces in traceVector :4528
Total Hops :65241
Average Hops :14.408348
Unique hosts (incl. Testboxes) :3962

Will now create graph from trace data
collecting nodes: 3962/3962
collecting links from trace: 4528/4528
DONE!
Nodes in graph : 3962
Links in graph : 7365

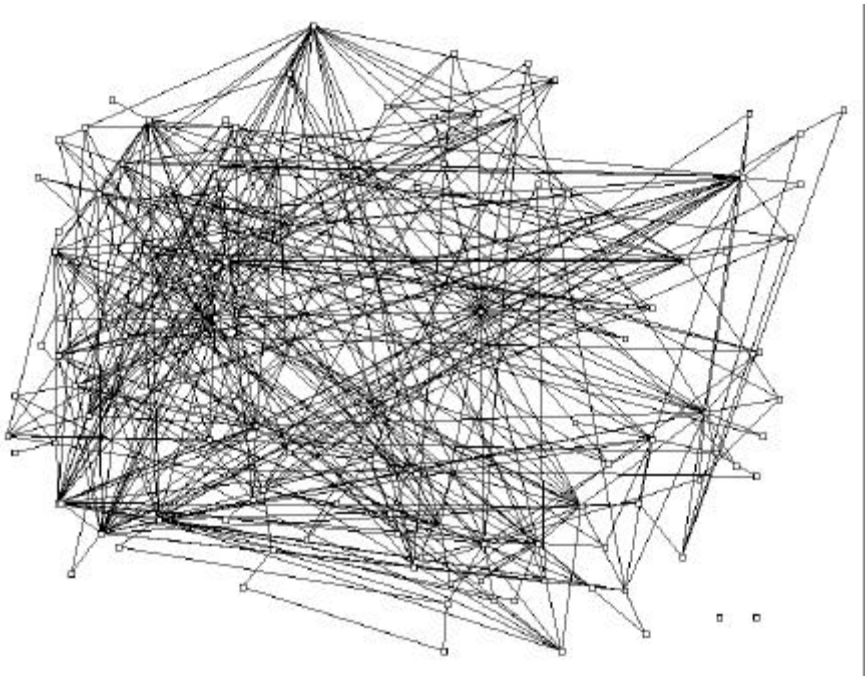
Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):combinedyyyIP.odf
Writing ODF file:
Writing nodes: 3962 written!
Writing links: 7365 written!
DONE!
[r]ead hashtable from file or create new hashtable [any other key]?
r

Will now try to map al the unique hosts onto an asn:
Host 3962 of 3962 mapped :3752 failed: 210
Writing hosts + mappings to : hosts.txt

Will now create a reduced graph from trace data
collecting nodes: 130/3962
collecting links from trace: 4528/4528
DONE!
Nodes in graph : 129
Links in graph : 430

Will now export Graph to ODF file
Enter filename (will place in .\data\odf\):combinedyyyAS-.odf
Writing ODF file:
Writing nodes: 129 written!
Writing links: 430 written!
DONE!

Isolated nodes: 8284 en 3228



AS graph of combined dataset with unmapped nodes removed

Appendix F. The testboxes

```
#############################################################################
#This file contains the testboxes in the Project                            #
#note : Parser is very simple, so be sure to follow this format exactly     #
#Format: [name] [ipaddress] [latitude] [longitude] [description]            #
#note : Only description can contain spaces                                 #
#Note : No blank lines are allowed (make sure of this!!)                    #
#############################################################################
tt01.ripe.net 193.0.0.4 52.3728 4.8878 RIPE NCC,Amsterdam,NL
tt02.ripe.net 193.0.0.2 52.3726 4.8880 RIPE NCC,Amsterdam,NL
tt03.ripe.net 193.0.0.28 52.3728 4.8878 RIPE NCC,Amsterdam,NL
tt04.ripe.net 193.0.0.6 52.3728 4.8878 RIPE NCC,Amsterdam,NL
tt07.ripe.net 192.36.143.148 59.3176 18.0231 Stupi,Stockholm,SE
tt08.ripe.net 193.10.252.21 59.3491 18.0691 NORDUnet,Stockholm,SE
tt09.ripe.net 195.67.159.40 59.3306 17.9630 Telia Network,Stockholm,SE
tt10.ripe.net 195.92.167.7 53.8023 -1.5343 Planet Online,Leeds,UK
tt11.ripe.net 192.109.251.32 48.1505 11.6200 ECRC Network Services,Munich,DE
tt12.ripe.net 193.149.44.25 48.1851 11.6015 Spacenet,Munich,DE
tt13.ripe.net 192.87.106.111 52.3561 4.9531 Surfnet,Amsterdam,NL
tt14.ripe.net 164.128.138.66 47.3900 8.5151 Swisscom,Ittingen,CH
tt15.ripe.net 194.179.3.155 40.4036 -3.6953 Telefonica Data,Madrid,ES
tt16.ripe.net 195.249.96.237 56.1081 10.1405 Tele Denmark Datanet,Viby,DK
tt17.ripe.net 193.43.2.18 44.4861 11.2601 CINECA,Bologna,IT
tt19.ripe.net 192.115.187.100 32.6306 35.0675 Golan Consulting,Haifa,IS
# tt19 heeft nu 212.116.184.53 als ip
tt20.ripe.net 195.178.64.60 50.1015 14.3916 CESNET,Prague,CZ
tt21.ripe.net 194.122.227.118 49.0126 8.4236 Xlink,Karlsruhe,DE
tt22.ripe.net 194.160.23.2 48.1515 17.1131 SANET,Bratislava,SK
tt23.ripe.net 209.211.237.18 41.1160 -73.7096 Advanced,Armonk,US
# tt23 had in 07-2000 als ip 207.24.7.8: truukje:
tt23.ripe.old 207.24.7.8 41.1160 -73.7096 Advanced,Armonk,US
tt24.ripe.net 212.70.195.122 37.9765 23.7340 TEE,Athens,GR
tt25.ripe.net 194.125.76.6 53.3343 -6.2573 Ireland Online,Dublin,IE
tt26.ripe.net 195.66.248.74 51.4968 -0.0188 Linx,London,UK
tt27.ripe.net 38.159.59.4 30.3195 -97.7006 MIDS,Austin,US
tt28.ripe.net 134.79.196.38 37.4175 -122.2038 SLAC,Stanford,US
tt30.ripe.net 212.56.224.2 49.6936 6.3288 Astra.Net,Luxembourg,LU
tt31.ripe.net 192.65.185.39 46.2326 6.0463 CERN,Geneva,CH
tt32.ripe.net 130.192.68.200 45.0400 7.6500 CSP,Torina,IT
tt33.ripe.net 193.194.137.254 52.0205 8.5368 Dev.Consulting,Teuto,Bielefeld,DE
tt34.ripe.net 193.166.4.105 60.1838 24.8173 FUNET,Espoo,FI
tt35.ripe.net 193.1.198.10 53.3381 -6.2403 HEAnet,Dublin,IE
tt36.ripe.net 129.88.9.1 45.1936 5.7700 IMAG,Grenoble,FR
tt37.ripe.net 195.112.71.35 47.3813 8.5168 Internet Acces,Zurich,CH
tt38.ripe.net 213.254.0.6 45.0728 7.6820 ITGate Network,Torino,IT
tt39.ripe.net 62.157.150.94 51.3541 8.2510 MEGLA Gmbh,Meschede,DE
tt40.ripe.net 212.5.128.15 42.6558 23.3708 Mobikom,Sofia,BU
tt41.ripe.net 195.185.187.44 50.0323 8.7051 Nacamar,Dreieich,DE
tt42.ripe.net 194.177.210.214 37.9856 23.7335 NTUA,Athens,GR
tt43.ripe.net 193.158.32.29 48.1896 11.6505 Prosieben Info Serv,Munich,DE
tt44.ripe.net 205.168.101.105 39.6825 -104.9691 Qwest,Denver,US
tt45.ripe.net 193.216.127.6 59.9243 10.8106 Tele2,Norway,NO
tt46.ripe.net 129.250.33.231 39.5713 -104.8808 Verio,Denver,US
tt47.ripe.net 130.217.248.132 -37.7888 175.3180 New Zealand
tt48.ripe.net 212.3.192.39 32.0593 34.8486 Surfree Ltd,IL
tt49.ripe.net 157.159.11.30 48.6246 02.4430 INT,Evry,FR
tt52.ripe.net 213.136.0.252 52.3065 05.6250 Business Internet Trends,NL
tt05.ripe.net 193.0.0.24 52.3728 4.8878 RIPE NCC,Amsterdam,NL(weg)
```