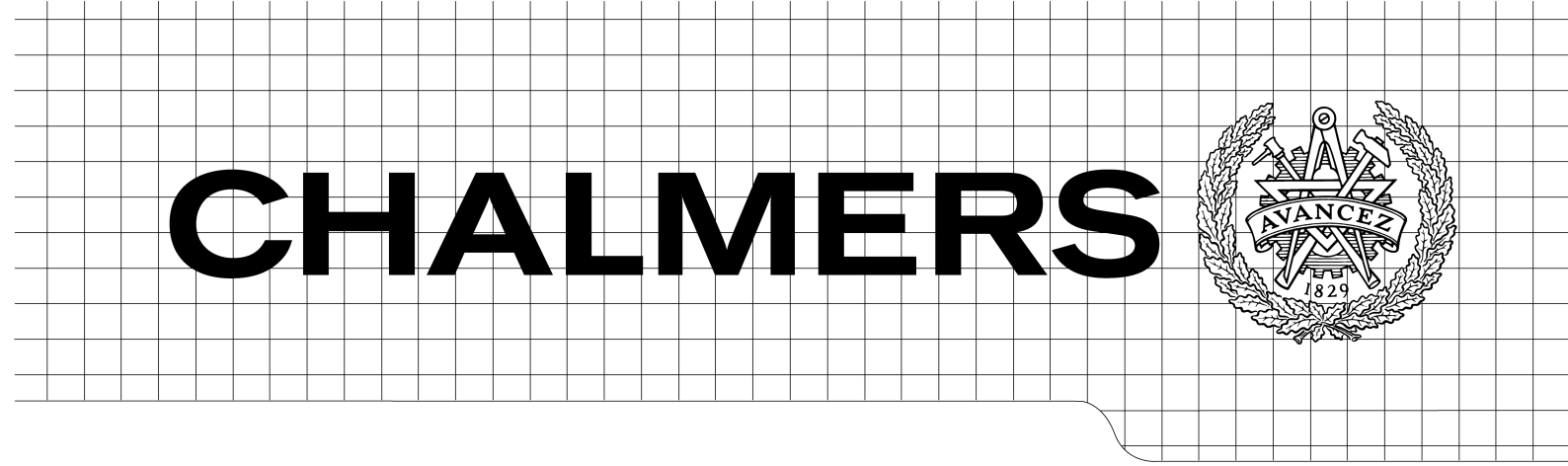


MASTER'S THESIS 2010

Computer Network Analysis by Visualization

Pauline Gomér and Jon-Erik Johnzon

Department of Computer Science and Engineering
Division of Networks and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2010

Network Analysis by Visualization

PAULINE GOMÉR
JON-ERIK JOHNZON

Examiner: TOMAS OLOVSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden

Contents

Abstract

The explosive growth of Internet has raised interest in traffic analysis. Understanding what traffic traverse the network is important for operation, investments, research and design of new protocols.

However network traffic analysis has not evolved as rapidly as network usage. Many researchers still look at data in raw text format even though the human brain is much better at pattern recognition in images than text. The purpose with this thesis is to develop a tool that builds a graph to visualize network traffic. Network analysis using this approach is not new, but there are no tools available where visualization is the focus.

The network graph is built by defining hosts as nodes and communication between hosts as edges. To enable analysis the user can select a subset of the traffic to visualize. The tool is able to produce graphs on large data sets, 3000 nodes and 30000 edges, on a home computer. We have tested the tool on generated data and on data provided by the MonNet project.

The tool is ready for testing but further development is needed since the graph library we used is resource intensive when visualizing large graphs.

Preface

This master thesis of 30 credits concludes our studies on the Software Engineering and Technology master programme at Chalmers University of Technology.

We would like to thank Wolfgang John and Tomas Olovson for their support during our work.

1 Project Goal

The main objective of this project is to develop the ideas presented in a paper by Iliofotou et al. [15] and produce a tool for visualizing network traffic. Iliofotou et al. proposes to visualize network traffic as Traffic Dispersion Graphs with IP addresses as nodes and communication between two IP addresses as edges. Different protocols will appear as different patterns in the graph.

This tool should not be specialized towards specific kinds of traffic. The user should be able to filter the traffic and visualize the result as a graph. We present a few examples of possible use cases in order to show that the software works as intended and give an indication of its usefulness. When starting the project, we had three goals:

- The tool should be able to take traffic traces as input and produce graphs and numerical graph features as output.
- The tool should include additional features like different (pre-) filter options, zooming capabilities, information retrieval possibilities per node (on click or on-mouse over).
- We should also provide examples that shows that our tool work as intended.

2 Introduction

The Internet has evolved beyond what anyone would have guessed. It has moved from being a tool for researchers and experts to a social and economical platform on which many build their lives. The reasons are many but the biggest are ease of sharing information, near instant communication and inexpensive usage.

However, the architecture and fundamental protocols used on the Internet have remained mostly unchanged. This presents problems for network operators and users since its difficult to understand how protocols can be used or misused. The aim when the Internet was created were to make communication work seamlessly between networks with different types of hardware. Important properties were the ease of including new networks and users, routing traffic through intermediate networks regardless of content and using a common addressing scheme [6]. Many of the problems on the Internet come from abuse of design decisions. This has raised an interest in analyzing and monitoring traffic on the Internet.

2.1 Motivation

Analysis of network traffic is important for operation, research and development of networks and design of new protocols. For these reasons we have developed a visualization tool to make traffic analysis faster and easier for humans.

2.1.1 Operation

When the number of users grew faster than network capacity, network operators faced the challenge to keep the network running smoothly while scaling it to support the increasing network traffic [3]. In the 90s, the growth was handled by expanding and upgrading the infrastructure. This was possible by venture capital that flowed into the new Internet sector and generated investments in the infrastructure. However, when the speculative bubble burst in the early 21st century, the network operators could not afford to continue upgrading as before and needed to make the most of existing capacity.

To optimize usage of the available capacity, network operators try to prioritize some traffic over other by using different Quality of Service (QoS) methods. QoS works by distinguishing traffic belonging to different traffic classes, such as sensitive, best effort and unwanted traffic. Each class of traffic is then treated differently; sensitive traffic is given top priority, best effort gives no guarantees and unwanted traffic is often blocked or limited [22].

QoS methods depend on accurate analysis methods to reliably classify traffic into the right classes. The Internet Protocols (TCP and IP) use addresses and port numbers to deliver the traffic to the right host and application. While Internet Assigned Number Authority (IANA) [14] lists many ports as belonging to specific protocols[1], the port numbers are not enforced and a protocol can use any port number for its traffic. The analysis methods must be able to distinguish traffic belonging to different application protocols even if they are using the same port number. A false positive or negative in a live environment to implement QoS could result in blocking of sensitive traffic or priority to unwanted traffic if the analysis results are used in routers to decide which packets to discard when there is congestion.

As the Internet gained popularity, it increased the incentives for users to make profit or gain other advantages by attacking other users [3]. A larger number of users makes it easier to hide and avoid detection as well as providing more targets. Malicious users generally aim to reach as many users as possible in exchange for a small success percentage. The attack methods often abuse or misuse features of protocols. An example of this is 'Ping of Death' where ICMP was misused to cause a buffer overflow in susceptible systems when a packet which exceeded the maximum size was sent fragmented, and when the receiver assembles it, the system crashed due to a buffer overflow.

Unwanted traffic does not really have any definition since it depends on who is doing the classification. Some network operators consider bandwidth intensive applications to belong to the unwanted traffic class. Streaming media is entering the scene and it uses a lot of bandwidth. The applications and protocols used are designed to use any available bandwidth to enable as many users as possible to use the service. Notable applications in this area are Spotify [29], Voddler [31] and Skype [28].

The network operators rely on statistical multiplexing[2] when dimensioning their networks to fulfill QoS constraints while minimizing unused bandwidth [25]. Statistical multiplexing assumes that the users most of the time are using less than their full bandwidth and QoS constraints can be met with less bandwidth than the sum of all users' maximum bandwidth. However, users that run bandwidth intensive applications do not follow this pattern and can cause congestion[3] and cause delays for all users.

On corporate and privately owned networks, the network operator might

[1]The TCP and UDP protocols allow 65536 ports. 0-1023 are for common system services and are called well-known ports. 1024-49151 are for common applications and are called registered ports. The rest of the ports are intended for dynamic and private ports.

[2]Multiplexing: Several users are sharing a resource.

[3]There is more traffic than the network has capacity to handle.

also block other kinds of applications due to policies or competition with their own solutions.

2.1.2 Research and development

The composition of the traffic on the Internet is constantly changing. The changes are usually relatively small, but every now and then a new protocol grows to dominate the network.

The small changes are caused by new versions and replacements of existing protocols and applications. In HTTP[4] 1.0 a new connection is created for each object that the web browser needs to fetch which results in many connections that only last until the transfer is done [2]. With HTTP 1.1 persistent connections was introduced and all objects on a web page are transferred over a single connection which results in fewer but longer transfers [10]. Other changes come from the introduction of new applications with same functionality as existing applications. One example is the development of file sharing. The first file sharing hubs were using IRC to connect, from this Napster came which evolved the IRC idea and made it easier and more intuitive. After this KaZaA, Morpheus, E-Donkey and Gnutella entered the scene, a short while after Direct Connect and it's derivatives reformed the P2P world and in the early 00s the BitTorrent protocol revolutionized the file sharing community.

The dominating type of traffic on the Internet has varied over time and the shift over to the newer protocol was very abrupt. Before the middle of the 90s, the traffic consisted mostly of email and file transfers [35]. Then the HTTP protocol was released and grew to make up most of the network traffic. Around the beginning of the 21st century, HTTP lost ground to various types of P2P traffic [3], and today P2P traffic is being replaced with streaming services.

Each type of traffic has different demands on the network. File transfers usually consist of long sessions with a lot of data and depend on the capacity between hosts. Web traffic has its bottleneck at the servers since they need bandwidth and resources to answer all requests from clients. P2P applications generate small packets, to keep the contact with all peers, with occasional data transfers between peers and are designed to make the most efficient use of each peer's bandwidth. Streaming services need high throughput but is less affected by latency since the receiver builds up a buffer before showing the data.

An optimal network configuration depends on the dominating traffic type,

[4]Hypertext Transfer Protocol

and that traffic type needs to be identified before any configuration optimizations can me made. With the help of traffic analysis on live traces, network usage can be reliably established.

2.1.3 New protocols

Designing protocols is not a trivial task. All consequences of design decisions cannot be known in advance as the protocol might be used in a situation or way that was not considered. A common difficulty is to foresee possible ways to misuse the protocol.

Design flaws in the protocol specification might not be discovered until after it is finished and released. The Wired Equivalent Privacy (WEP) protocol was designed to secure wireless networks and was used for several years before serious flaws in its design were discovered [11]. Further study of the protocol revealed that under certain circumstances the protocol could be broken in less than one minute [30].

In addition, implementations of the same protocol may behave differently yet still conform to specification. The differences comes from how the implementor interpreted the specification and deals with imprecise details. In a comparison of the behavior of implementations of TCP, several differences were detected with at least one violation of the standard. [5].

One conclusion that can be drawn from this is that it is always necessary to look at what others have done when designing something new, and that even though something looks like it is working correctly it might not be. Mistakes in the implementation or the protocol itself can be found by analyzing the protocol's actual behavior on the network; the new protocol might be suppressing other protocols.

2.2 Issues with traffic analysis

2.2.1 Collecting data

Collecting packet traces from operational networks is important for research and development. While traffic traces from closed, controlled networks have their uses, they cannot replace packet traces with real traffic data.

However collection of data brings up several integrity and privacy concerns. The content of the packets can reveal sensitive information and even if the payload is discarded, the data is enough to create sociograms and trace how hosts interact.

2.2.2 Data amount

As the volume of traffic in the networks grows, the amount of storage needed for collection also increases. There are several possible approaches to reduce the size of traffic traces. However they all are tradeoffs between completeness and scalability [1].

Complete packet-level traces is an approach where all information in all packets is kept. It provides the most complete record, but does not scale well due to the large storage requirement.

Trace reduction techniques discards or summarize some of the information contained in the packages. Here are three common techniques:

- Analysis of SYN/FIN/RST packets takes advantage of that a lot of information about a connection can be found in the TCP control packets[5]. By only collecting the control packages, much less storage is needed as no data is kept. The drawback is that information about the application and its protocol is not preserved.
- Unidirectional traffic analysis takes advantage of that the TCP protocol provides reliable data transfer in both directions. That makes it possible to determine useful information about the data flow in the unseen direction while only needing to collect half the number of packets.
- Packet and flow sampling only collects a small portion of the total traffic such as one packet for every N packets or flows with more than N packets. This approach only provides general knowledge about the traffic traces such as packet distribution and inter-arrival times.

Problems There are several problems with collection of packet traces. But the biggest is that it is impossible to start a trace and collect 100% of all communications. There are packets that are missed because the link was established before the packet collection started or there are connections that use another link for part of the communication.

2.2.3 Analysis of data

Obfuscation is a general term for any method that tries to prevent an application's traffic from being correctly classified. A common approach is

[5]SYN packets request a new connection. FIN packets closes the connection. RST packets refuses the connection or initiates a teardown of an existing.

to use ports that either is associated with another application or use a random port. Encryption is also used to hide application headers in the payload that can be used to identify the application.

Unknown protocols will always make up a portion of the traffic traces as there are constantly new protocols and applications introduced on the Internet.

2.3 Overview of classification methods

Classification of network traffic has become a challenging problem. As the methods for analysis and classification are also used for filtering and blocking, users are continuously seeking new ways to obfuscate their traffic.

This race between network operators and users drives the development of new analysis methods. The oldest methods were very naive and relatively easy to counter as they relied on exact matching. Newer analysis methods try to classify traffic by studying its actual behavior.

2.3.1 Exact matching

Analysis methods that use exact matching inspect each packet or flow and classify it based on the information it contains.

Port-based analysis Traffic classification based on port numbers is one of the earliest approaches to identify traffic. The port number is used together with the IP address to define the endpoint of a communication flow on a network. It is the fastest and simplest approach to classification [21].

However, network operators began to limit or block traffic based on ports and the affected applications started to use other applications' ports or generate a random port for each user. By using ports associated to other applications, the traffic would be misidentified. By using a random port, the traffic is difficult to identify as belonging to one specific application, additionally it interferes with classification of other protocols.

Signature-based analysis The classification of traffic is done by inspecting the packet's payload and looking for a match against known application protocol signatures. A protocol signature is only a string with a pattern that usually occurs in packets belonging to that protocol. The first data in the payload is typically application headers that are fixed. As each application has its own headers, this string of data is seen as a unique signature and could be used for classifying the traffic.

The signature-based analysis methods have several problems [9]:

- The protocols and their signatures must be known in advance. The signatures are generated from samples of the protocols. Therefore they cannot be used to classify unknown protocols.

- Keep the signature database up to date. Many applications are still under development and their signatures must be regenerated and added if the protocol headers are changed. This requires constant effort to keep track of new releases of a protocol and check if the signature are still valid. With a large number of signatures to keep track of this results in a lot of extra work.

- Fails when the payload is encrypted. As the signature is a fixed pattern, the matching does not work if the payload is encrypted. A signature "GET" matches the beginning of "GET http://www.wikipedia.org", but a simple shift cipher such as Caesar's cipher [32] will turn that string into "JHW kwws://zzz.zlnlshgld.ruj"[6] and the signature will not match.

2.3.2 Machine learning

Machine learning is a discipline where a computer learns from training data and later use that knowledge to make decisions about real data. The training data is a set of examples that contain the sought data. The computer uses a clustering algorithm that tries to find patterns in the examples. In network analysis, machine learning uses flow attributes such as connection duration and byte counts [24].

There are two types of machine learning approaches based on how the training data is provided: supervised and unsupervised learning [26].

Supervised learning In the supervised learning approach the training data is already classified. The data set contains traffic traces that has been pre-classified manually.

The learning machines's task is to find common patterns among the traffic flows and derive rules to identify the pattern. The rules that are produced are then used to classify new traffic data [26].

The main drawbacks are that the resulting rules are limited to the classes and patterns that are in the training data [9]. Patterns that are not in the

[6] As Caesar's cipher relies on the letters position in the alphabet, punctuation marks and other non-letters are usually excluded or ignored.

training data, but do belong to one of the classes, might not be correctly classified if they differ too much from the other related patterns.

Therefore, supervised learning is used when the aim is to single out specific traffic classes.

Unsupervised learning In the unsupervised learning approach the computer is given unclassified training data. The learning machine does not get any information in addition to the traffic traces and instead relies on heuristics to group the data.

The learning machines's task is to find common features among the examples and clustering those together. There are three basic clustering methods that can be used [26]:

- The classic k-means algorithm that puts each example in exactly one group
- The incremental clustering method produces a hierarchical grouping where the examples are first placed in general groups at the top level which then become more fine-grained with each level
- The probability-based methods that calculate the probability of an example to belong to a group

Unsupervised methods can reliably classify encrypted traffic as well as new applications [9]. Since the algorithm does not use the packet payload, encrypted traffic can be correctly classified by identifying unencrypted traffic that is in the same cluster. New applications can be identified by studying which other application it is clustered with.

2.3.3 Heuristic methods

Heuristics can be used when the traffic traces do not contain packet payload and accurate training data is not available [17].

The analysis method uses a set of heuristics based on connection patterns. When analyzing P2P traffic [19] [27] [17], the traffic trace was first divided into intervals and then the heuristics models was applied to each interval. Possible P2P traffic is flagged if it matches heuristics that describe common behavior for P2P applications such as concurrent usage of TCP and UDP. Heuristics was then used on the flagged traffic to remove false positives from protocols with similar behavior such as DNS, mail, and web traffic [17].

The heuristic methods have several weaknesses. The length of the intervals affects how well the heuristics work. Longer intervals tend to give better

results, but the possibility that a host changes behavior in the middle of an interval increases [17]. The heuristics must also handle the possibility that only the traffic in one direction is available due to asymmetric routing [19].

On the other hand, heuristics can use port analysis to reliably classify common applications. While other applications obfuscate their traffic by using the same ports, the protocols they are using for communication generates patterns that differ very much from the original applications, especially packet size and TCP headers [21].

2.4 Related work

2.4.1 Behavior-based network analysis

Behavior-based Network analysis is based on sociology and sociograms[7]. Using concepts from graph theory, the idea is to look at the network data and construct a graph where nodes represent hosts and edges represent some form of interaction. This is one of the big problems: how to define an edge, and there is a lot of discussion about this topic [15].

In the tool we have developed, we choose to draw an edge as soon as we see a packet between two hosts. This is a simple solution but can easily be changed. By having creating a graph from the network trace, we can easily calculate graph metrics, e.g. in and out degree of a node, edge weight, and then map these to network properties; in and out degree is trivially incoming and outgoing connections from a host, while edge weight can be a lot of different things, e.g. number of packets, number of protocols, and number of bytes transferred depending on what the user chooses.

While the concept of behavior-based network analysis isn't new, there is not any tool that emphasizes on visualization. Both Graption [15] and BLINC [20] uses behavior-based analysis but do not provide any visualization. This is where our tool comes in, to give the user a picture of the network.

Our approach combines previously used methods like well-known ports, protocol numbers (assigned by IANA [14]) and IP prefixes. In addition we add metrics such as indegree, outdegree and number of packets and bytes. These properties gives our tool the ability to provide a deep and fine-grained analysis on large traces.

[7] Who knows who and who communicates with whom

3 Requirements Specification

3.1 Software

When planning our project we agreed on a list of requirements on our software tool and the operation environment. This list contains both functional requirements and non-functional requirements.

3.1.1 Functional Requirements

- Be compatible with as many trace formats as possible
- Provide a GUI
- User defined filters

Trace format compatibility There are a lot of different formats that can be used to store network traces, for example dag and pcap. To make the application compatible with as many formats as possible we chose to use CoralReef [7]. CoralReef is a de facto standard tool to analyze network traces. The functionality we use is called crl_flow which converts network traces from binary packet data to flows in text format. CoralReef is also free and open source, however CoralReef is only available a UNIX and UNIX-like operating systems.

3.1.2 Non-Functional Requirements

- Dependency Minimization
- Intuitive Interface
- FSF style license
- Run under Linux
- As resource-efficient with regard to memory and CPU as possible

Dependency Minimization This topic is a little unclear as minimization have two meanings: Use as few dependencies as possible and reduce the strength of the bonds between an application and its dependencies.

The first meaning refers to when writing everything from scratch instead of using libraries. Importing functions from libraries will not only include those functions, but also the functions they use which all add to the size

of the program. Therefore, the program can become noticeable smaller by avoiding importing functions.

The latter meaning refers to avoiding versions dependencies, that is to avoid requiring the latest version of a dependency unless the latest functionality is really needed. This is because some system run old software and upgrading one program will usually require its dependencies to be upgraded as well.

Our focus was to reduce the amount of external third party applications and libraries as much as possible. The reason was to make the application easy to install and deploy since installation and configuration of dependencies often require administrative privileges.

User Prerequisites

- Good knowledge of networks and protocols
- Know SQL

In addition to good knowledge of networks and protocols which is a natural prerequisite for this type of application we require that the user knows SQL, at least basic select statements.

This is because the application relies on SQL queries; the more skilled user is the better the application will performaed. That said the application is still usable even if the SQL knowledge is limited to simple select statements, but the application features and performance will be limited. But to someone who knows SQL or is willing to put some effort into it, there should be no problem writing and good queries.

Another requirement to get a good result is user knowledge. This is why we included networks and protocols in the prerequisites; it should be an obvious requirement for this application but we included it to make it perfectly clear that our tool isn't magical in any way. It is simply not possible to have it search for the latest P2P protocol without having a clue of how it works.

Licensing Source code for computer programs are in many countries defined as literary works and falls under copyright law. While copyright varies between countries[8], the right holder is typically granted exclusive rights to selling, distribution, and adapting their work.

Our application should have a license that allows:

[8]Most countries in the world are signatories of the Berne Convention for the Protection of Literary and Artistic Works and provides similar protection.

- free redistribution
- free to modify
- free to use

These criteria is typically fulfilled by open source and free software licenses. They was choosen to allow the tool to be spread and used widely without requiring permission.

Since our application will use third party libraries, a potential problem is that those libraries have different licenses that cannot co-exist in the same application. Therefore the preferred licenses are those that do not impose restrictions on which license our tool uses, such as releasing our tool under the same license. Two common licenses that does not have such restrictions are BSD and LGPL.

Operation Environment

- Ubuntu 8.10 or newer
- CoralReef 3.8.4 or newer
- MySQL 5.1 or newer
- SUN Java 1.6 or newer

3.2 Hardware

We recommend that our tool is run on a computer with at least the following specs:

- Pentium 4 3.6 GHz
- 1024MB RAM

We list this to give a reference to our performance benchmarks.

3.3 Test trace

3.3.1 Data source

Our test data comes from packet captures on NorduNet (Nordic University Network) collected by the MonNet project. The data consists of a 20 minute bi-directional capture conducted on the 10th of April 2006.

3.3.2 Data properties

Data about the trace (with the payload stripped off):

- 772 694 unique IP addresses
- 4 325 248 flows[9]
- 95 748 961 packets
- 69 662 721 548 bytes
- 3.2 gigabytes on disk (compressed)

[9] A flow is all packages from a source IP address and port to a destination IP address and port during a certain time period [34]

4 Implementation

4.1 Development

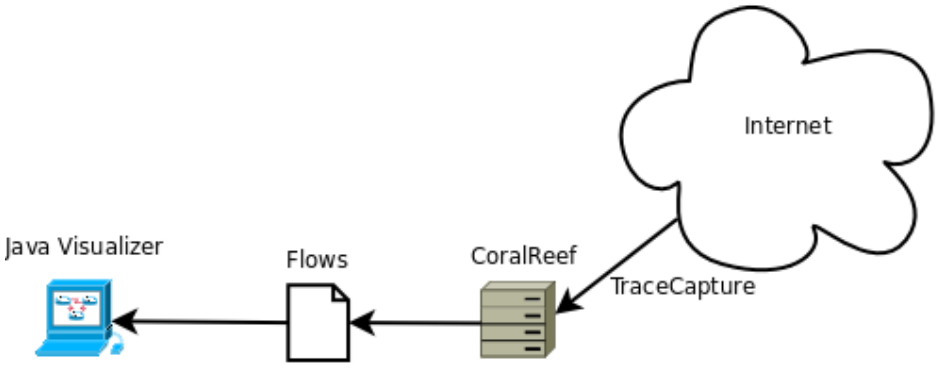
When we started to develop the application we had a set of base tools available. These were GraphViz [13], crl_flow (part of CoralReef [7]) and a Perl script to format crl_flow output to GraphViz input.

GraphViz is a software for visualizing graphs, but only produces output in various image formats. Since our goal was an interactive visualization of a graph which would be difficult with a static image, we decided to find another graph library.

We stayed with crl_flow because it's fast, customizable and supports many trace formats. So we wrote a parser for crl_flow output and started out trying different solutions for data storage and presentation. We tried different solutions here presented in chronological order.

- Java with plain text file storage (pts)
- Java with Haskell preprocessor and pts
- Java with Haskell preprocessor and MySQL storage
- Java with MySQL as both storage and preprocessor

4.1.1 Java with pts



This first solution was a Java implementation with no preprocessing. The only external tool we used was crl_flow to convert the trace to flows. Then we parsed the data to memory reading line by line and checking if each flow matched the current filter. When we tested this on a small sample, 5 seconds of our data set, it worked really well. Analysis and visualization was completed in a few seconds. Increasing the data set to the first 5 minutes

our application started to consume a lot of memory for more complex filters. For example an out- or in-degree filter on the nodes (which means that the application needs to keep each encountered IP address in memory with counters) memory usage was about 900MB. Although there are more efficient data structures (for example a Bloom Filter [33] could be used) than the Java ArrayList, it still does not scale with increasing traffic volumes.

Pros

- Lightweight
- Fast on small traces

Cons

- Bad performance on larger traces
- Heavy memory consumption

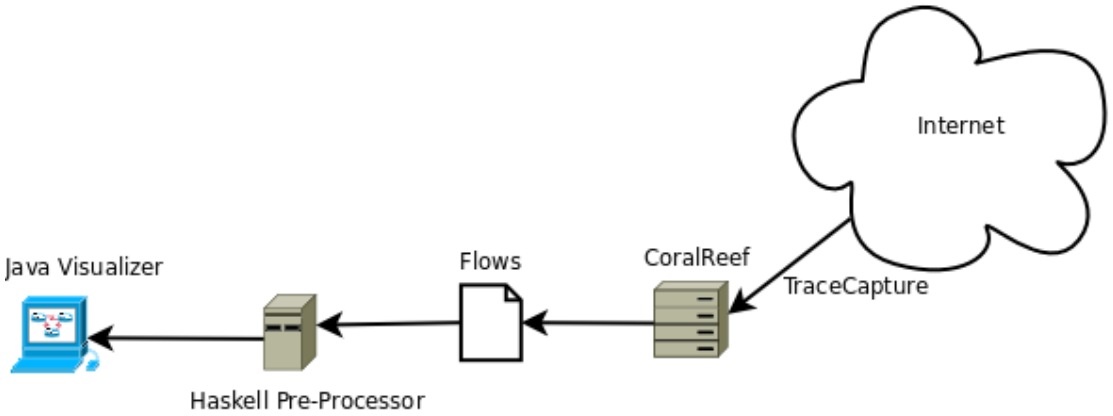
Lessons Learned Doing all computations at run-time gives poor performance since the application has to compute all statistics on data. It works well for small data sets but as soon as the set grows the performance drastically drops. The reasons for this are two: the calculations requires too much cpu time and all information must be kept in memory. When we ran the test with this approach we did not have a graph drawer, we just placed the data in we wanted to draw in memory. The conclusion drawn from this is that we need to pre-calculate some statistics using a preprocessor.

Preprocessor Clever data organization and storage are keys to get good performance, avoiding exhaustive search is vital. Using a preprocessor and store the results in a good way on a persistent storage is very important. We don't want to waste memory and CPU since it is needed for the visualization, as we are using Java memory management is even more important as to avoid runtime failures such as stack or heap overflows. The preprocessor will for each node calculate:

- Total number of packets received or sent
- Total number of bytes received or sent
- Total number of incoming connections
- Total number of outgoing connections

By using precomputed statistics, memory usage is minimized at runtime because the data is read directly from storage and does not need to be kept in memory. This type of optimization is used to gain execution speed by sacrificing persistent storage space, which is usually much cheaper than memory and faster CPUs.

4.1.2 Java with Haskell preprocessor and pts



The first failure made us think about how we could reduce memory requirements without reducing functionality. We realized that pre-computation was necessary to avoid storing the entire IP list in memory. To implement the preprocessor we needed a language which had good libraries for parsing and pattern recognition. Having taken some functional programming courses we chose to write the preprocessor in Haskell using the highly optimized ByteString library [8].

Now when we were using non-runtime calculations we chose to precompute all statistics we thought usable. Nodes had total amount of packets, bytes and flows sent or received summarized as well as their in- and out-degree computed. Edges however did not have much data computed since we could not find anything of value to compute. We also converted the IP addresses from the 4x8bit integer (x.x.x.x) notation to 32bit integer. The visualizer was rewritten to work with the above changes and performance increased, now we applied our filters on the complete 20 minute data set.

Performance for basic filters was very good, filtering out all traffic with a destination port 80 or all nodes with more than 50 000 bytes sent or received was completed in a few minutes. However, the performance of combined filters was still bad, taking close to an hour. An example of a combined filter is "nodes with out-degree >50 and communicating on port 80" and the slow speed of the combined filters along with the problem of heap overflow due to filters matching too much of the data set made this solution non-useful.

Pros

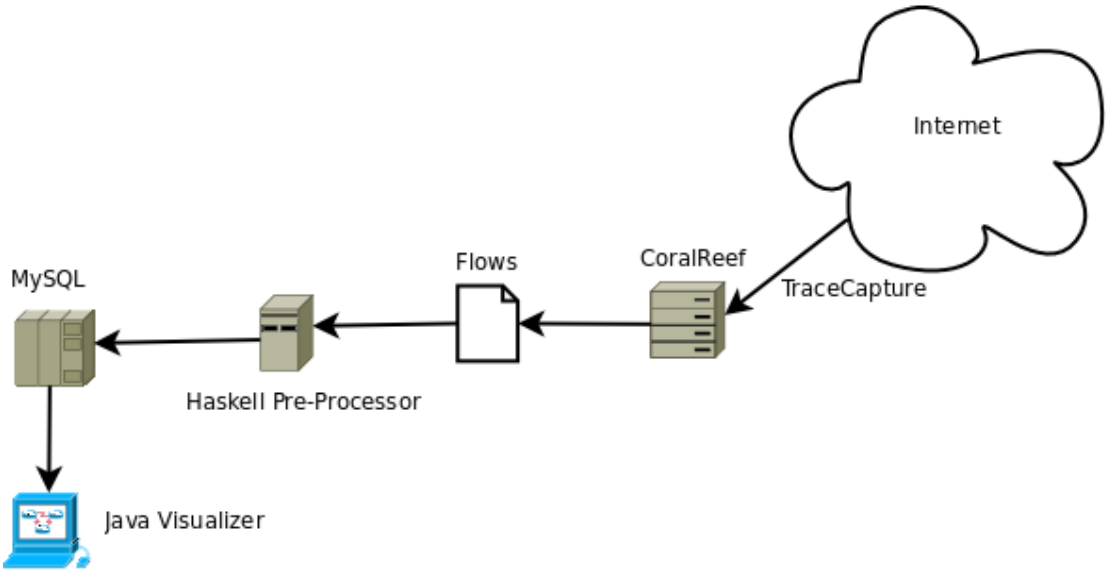
- Good for basic filters
- Good performance even on large traces

Cons

- Bad performance more complex filters
- Still heavy memory consumption
- Still does not scale

Lessons Learned Working with plain text storage seemed to be the most limiting factor. It required us to parse all the data and keep too much in memory, which means that the application had trouble working with traces as they grow larger since there is no way of figuring out how large the data set will be before testing it. The conclusion drawn from this was that we needed a database.

4.1.3 Java with Haskell preprocessor and MySQL



In an attempt to speed up the performance of the combined filters we decided to switch from the file based storage to a database. The choice of MySQL was made due to that both of us had experience from working with it before and that it is licensed under GPL. We still used the same preprocessor

but now instead of using the text files directly we created a table for each (one for nodes and one for edges) and then loaded them into the database using the LOAD DATA directive.

The switch to database storage led to a huge performance improvement. The basic filters where now completing in seconds instead of minutes and more complex filters were now done in minutes instead of hours.

However now we discovered a problem with the application. Now when we could test the application properly we realized that we allowed more than one edge between two nodes. While this may not seem like a big problem it rendered the graphs useless since there could be hundreds of edges between two nodes and it clutters the graph to incomprehension. To solve this problem we placed a unique constraint[10] on the edges in the database and updating the information on the edge when a duplicate was detected. This solved the multiple edge problem but at the expense of data loss since we could no longer distinguish the individual flows between two nodes. Information loss in an analysis tool is not acceptable since we don't have a clue to what data our users need.

Pros

- Good for basic and complex filters
- Good performance even on large traces
- Better scaling
- Good memory management

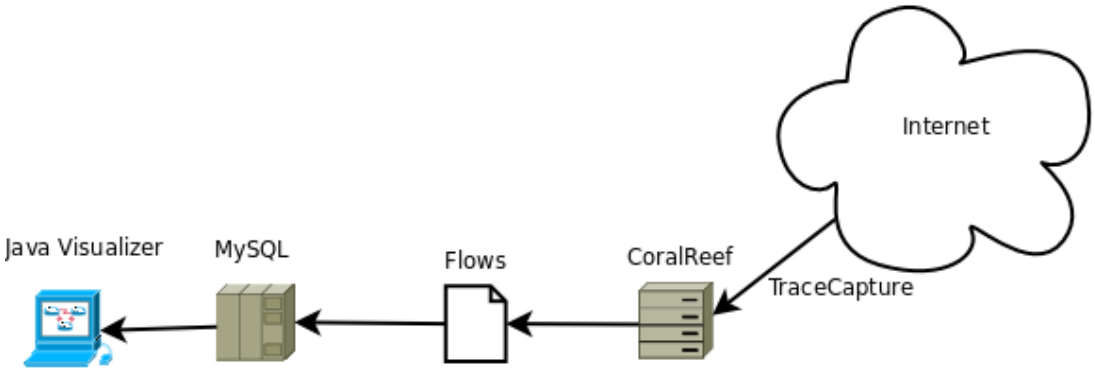
Cons

- Information loss!

Lessons Learned This solution seemed to solve all problems we had. It relieved the memory appetite, gave good performance, enabled size estimation. But one severe bug appeared, the way we added the data into the database made us loose data about which protocols were responsible for what traffic. This isn't good because that information is useful when searching for malicious traffic. The problem resided in our preprocessor and after much effort to fix it we decided to drop our preprocessor. We first considered writing a new one in C or the like but we realized that we could use MySQL constructs instead, more specifically triggers and on-update.

[10] A unique constraint is used to guarantee that no duplicate values are entered in specific columns which are not part of the primary key

4.1.4 Java with MySQL



To solve the data loss and multiple edge problems we decided to make a proper database design.

We also found a way to make the statistics computations during the inserts into the database, using triggers[11].

This table setup minimizes redundancy and solves a visualization issue when there is more than one flow between two hosts. This table layout creates an edge as soon as some form of communication has been seen between two hosts. Then the actual data is stored in the edge-data table, along with the corresponding edge id. With the edge id in the edge-data table, the actual edge can be retrieved and from the edge to the source and destination nodes via the IP addresses.

Timestamps and traffic volumes are stored in the edges table to allow searches for high bandwidth and/or high throughput links. The reason for storing traffic volumes in the nodes is to be able to search for traffic heavy nodes.

Pros

- Good for basic and complex filters
- Good performance even on large traces
- Better scaling
- Good memory management
- All information intact

[11] A SQL trigger is a statement that is activated when some action triggers it, hence the name. This event can be before an insert or after an update for example. When the trigger is triggered some SQL statement is executed to perform some action that the database creator wanted.

Cons

- Reliance on triggers

Lessons Learned This solution fixed the severe issue with the previous solution while keeping all of the improvements. But it added a rather big constraint on the database, it must support triggers. This means that MySQL version 5 or later is required, as well as having a user account with the SUPER privilege on the database. However, the SUPER privilege is only required to add data to the database, but not to use existing data.

4.2 Database optimization

With the introduction of the database the performance of the visualization tool improved considerably. Further performance improvement can be achieved by using good indices. Bad or no indexing forces the database to rely on linear search through the tables.

Linear search is rather slow on the data volumes we are using, however building good indices requires knowledge about the data and how its going to be used. A trivial approach is to just build indices for each column; this only improves performance for filters operating on that column only. To really optimize the performance of the database, indices over many columns are needed.

However these indices take time to build and increase the database size since there are many combinations (for a table with 8 columns there are 8! (40320) different combinations). But the vast majority of these indices will never or very seldom be used. To understand how to build good and useful indices, an analysis of common filters must be done.

4.2.1 Filter Analysis and Database Indexing

There are many trivial filters like source port and destination port. These are also trivial to build indices for, by building an index for the column to filter on. But more advanced filters can be built, for example to search for TCP-SYN scans on possible HTTP-servers, out-degree>300 and bytes <50 and protocol = 6 (TCP) and destination port = 80. To speed this search using indices we need the following: one index on the out-degree column in the nodes table and one index on the edge-data table columns protocol and destination port. The bytes column contains so many different values that an index will be very large, but this column could of course be added to maximize performance.

To optimize the indices for the IP-addresses, we store the each address as a single 32-bit integer, since indices work better on numerical types than strings. The conversion from standard doted IP notation is done with the MySQL function INET_ATON, for example INET_ATON('127.0.0.1') is 2130706433.

One thing to have in mind when constructing indices in MySQL is that the indices work like this, think linked-list:

1. col1 → col2 → col3
2. col1 → col2
3. col1

So if an index is built from col1 to col2 to col3, an index from col1 to col2 and an index on col1 is automatically gained, but no indices on col2, col3 and col2 to col3. This is useful because some indices are given for free when constructing large ones, and thus redundant indices are minimized [23].

4.3 Database Initialization

The database setup process is a one time per trace process. This process uses crl_flow to convert traces to flows, and then Stream Editor (sed) is used to format the output. This result is then loaded into our database using a shellscript.

Using crl_flow To get the needed output from crl_flow use this command: the following arguments are needed: crl_flow -Tf60 -o outfile tracefile

- -Tf60: this sets the flow timeout to 60 seconds. This means that a flow is terminated 60 seconds after the last packet is seen. The CAIDA website [4] recommended setting the flow timeout to 60s.
- if the input is a bi-directional trace contained in two separate files they can be merged by using the flag -m, i.e. -m tracefile1 tracefile2 instead of tracefile.
- -o specifies the output file name.

Stream Editor - sed Sed is used to format the flow data for parsing using a regexp to clean and format the crl_flow output.

- Remove all comments (lines beginning with #)
- Split the timestamp into seconds and subseconds[12]

[12]crl_flow denotes timestaps in seconds.subseconds (fractions of a second). This is more difficult to parse using a regexp since the IP addresses also contain dots. So we reformat the timestamps to list seconds and subseconds separately.

Database script This script creates the required tables in the database and fills them with data. It works by collecting all needed information over the command-line and then performs, in the listed order, these operations:

- Create and populate the protocol table if this does not exist.
- Create the node, edge and edge-data tables for this trace
- Create the triggers used to calculate the statistics
- Load the data into the database using bulk load

4.4 Graph Library Research

We needed a graph library for visualizing data as a graph in our application. When we searched for candidates we found three that looked promising JGraphT, JGraph and JUNG.

When we took a closer look we found that JGraphT did not have built-in visualization but instead provided interfaces to MSAGL and GraphViz. Neither of these were of any use to us since MSAGL is a commercial product for Windows and developing interactive graphs using GraphViz is cumersome.

The second library we looked at was JGraph. This library did not provide any built-in visualization either but the developer had another product that provided visualization called LayoutPro. Unfortunately the licensing was very unclear; there was an Academic License available but not much details on how it allowed the application to be used or distributed [16].

The third library we looked at was JUNG (Java Universal Network/Graph Framework). This library had both a Graph structure and a competent visualization engine with many layouts and filter options. An extensive API and many code examples were also available so that it was a low threshold to get going. JUNG is also free software licensed under a BSD license.

Among the three candidates only JUNG fulfilled all our requirements and we therefore chose to use JUNG in our visualization tool.

4.5 Application interface

The application interface consists of two parts: filtering and visualization. Filtering the data is necessary since the database can handle much more data than the application. It also enables the user to select a subset of the data for display. The visualization of the data is done as a graph.

4.5.1 Filter interface

In the first version of the application, no filtering was available through the interface. This version was primarily for testing the graph library. Skeleton menus were added in preparation for holding different graph manipulation operations.

At this point of time, we were using plain text storage and had not decided on how to apply the filters. The options we considered were to type the filter rules on the command line when starting the application or using a dialogue that would be accessed from a menu. If the filtering was done on the command line, we would have to write our own parser and filtering language. If we used a form in a dialogue, we would need to make it flexible enough to handle more complex filters.

In the second version, the interface was redesigned and the menus where discarded in favor of a tabbed layout. The filtering interface was given its own tab since the application needed to be able to handle a large database which contained far more information than it would be able to visualize. Therefore it is necessary to filter the input before drawing the graph. This made the filtering as important as drawing the graph and that was reflected in the new design.

The main reason for the redesign was that we replaced plain text storage with MySQL. MySQL uses SQL for querying and we could take advantage of its flexible and powerful syntax for filtering. We estimated it would be too time-consuming to create an interface that would give the user the same level of flexibility, power and freedom as SQL. This decision impose a new requirement on the user, but since the user would have had to learn a new filtering language in any case we chose to use SQL since there is a lot of material on it.

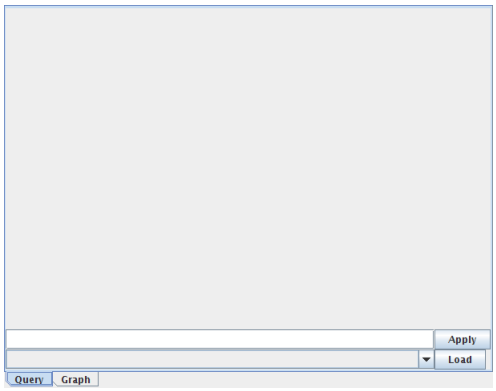


Figure 1: The query tab with query field

The filtering interface became a single text field that exposed the WHERE clause of a SELECT query[13]. A SQL SELECT query is responsible for retrieving data from a SQL database, the data retrieved is limited by the WHERE clause.

Example:

SELECT * FROM users WHERE username="testuser";

This would select the rows in the users database which has the username 'testuser'.

By exposing the WHERE clause the user does not have to write a full SELECT query and can focus on filtering the data.

In the third version of the application, the single text field became three fields with one for each table: nodes, edges and edge data. The SELECT query used in the second version was a join between the tables edges and edgedata as they contain most of the data. However, this left out the nodes table with statistics for each node since a three-table join would be too expensive. Instead each table was given its own field and the filters for each table narrow down the results.

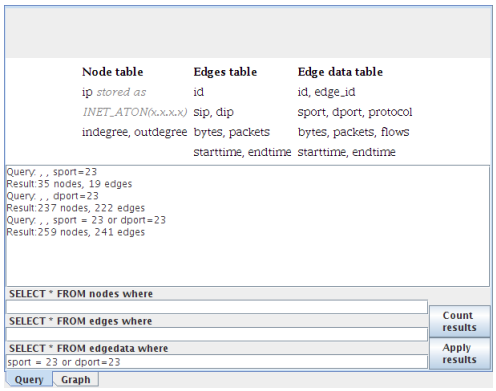


Figure 2: The query tab with three exposed MySQL queries and a log window.

New in this version was the description of the tables and the possibility to count the number of results from a query. Since there is a limit on how many nodes the graph library can handle depending on the hardware, we try lower the risk for a crash by providing the option of counting the number of results before drawing the graph[14].

[13]The WHERE clause limits which rows in the tables to include in the result

[14]Since it is impossible for us to know exactly how much data a computer can handle, there are too many variables to consider. We let the user estimate this instead.

4.5.2 Graph interface

The first version of the GUI was a prototype for testing the JUNG library. Since the focus was on the graph, a single window with the graph drawing area was used. In this version, the general graph drawing features were implemented such as scrolling, zooming and pick support (moving the whole graph).

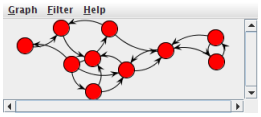


Figure 3: First version of the GUI with dummy data

The second version of the GUI used a tabbed layout with the filtering and graph drawing area on separate tabs. The application needed to be possible to use with far more data than the graph would be able to handle. That made filtering as important as the resulting graph and this was reflected in the design by putting the filtering before the graph view.

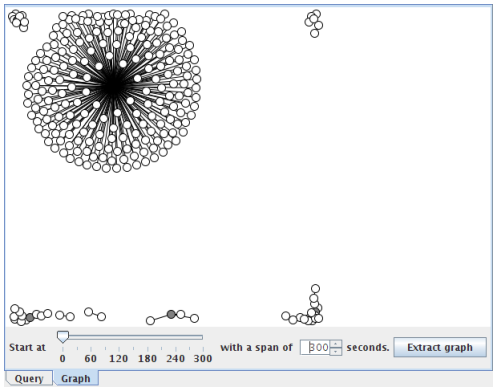


Figure 4: Graph showing traffic over the Telnet protocol (port 23)

Several new features were added to the graph tab: basic node information, time interval filter and extracting subgraphs.

Information about a node is shown in a popup window when it is clicked. The popup shows which IP address (in decimal notation) the node represents as well as in-degree and out-degree[15]. The degree values in the graph are based on the filtered subset that is used to draw the graph.

[15]Communication between two IP addresses may be limited to one direction. This can have several different reasons. Two plausible is that the trace is unidirectional the other

The filtering over a set time interval has two purposes: showing the change over time and de-cluttering the graph. If the initial filtering results in a large number of results, the drawn graph will be very cluttered and could be difficult to analyze. The time filtering will only show the traffic that occurs within the time interval while retaining the positions of the nodes.

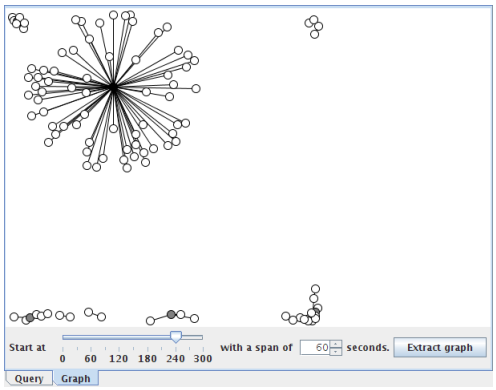


Figure 5: Only the traffic in the last 60 seconds of the time span is displayed.

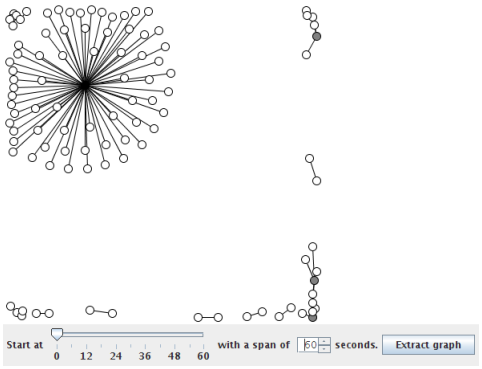


Figure 6: The sixty second time interval has been extracted to a separate graph and the layout has been recalculated.

The time filtering panel is placed under the graph and provides controls for showing the traffic in a smaller time interval. The slider marks the start while the spinner gives the extent of the interval. The interval is also used to define which nodes to include when extracting a subset of the graph. Since the time filter only controls which graph elements to display, the layout of the graph is unchanged and the placement of the graph elements appears unbalanced. However, the layout is recalculated when those elements are

is that the return traffic occurred outside the capture window. In-degree and out-degree is the number of incoming respectively outgoing connections.

extracted to a new window.

4.5.3 Graph Layout Algorithms

A good graph layout algorithm separates the nodes and minimize overlapping edges. We are using two layout algorithms: the Kamada-Kawai algorithm for the initial placement of nodes and the Fruchterman-Reingold algorithm for adjusting the length of the edges.

The Kamada-Kawai algorithm produces visually pleasing graphs with minimal crossing edges even for very large graphs [18]. While positioning of the nodes are good, the length of the edges are very uneven.

This is corrected by using the Fruchterman-Reingold algorithm which work on the principle of drawing connected nodes closer. The length of the edges between the nodes is calculated based on available space and number of nodes [12].

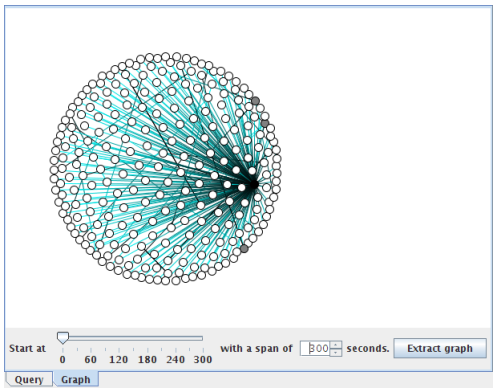


Figure 7: Using only the Kawada-Kawai algorithm.

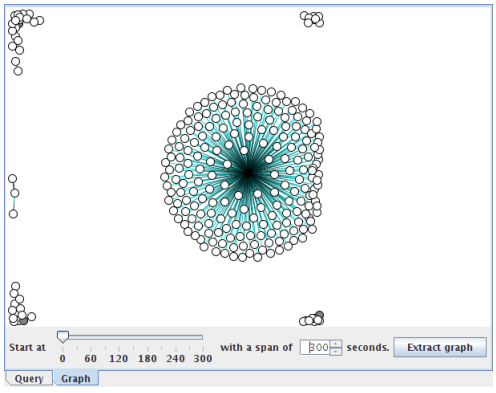


Figure 8: Using only the Fruchterman-Reingold algorithm.

5 Results

5.1 Application usage

5.1.1 Simple

A typical usage is to define a filter that only takes a small amount of information into account, this could be protocol, source port, destination port or a combination. The purpose of such a filter is to see if there is any traffic with those characteristics, for example is there any UDP traffic in the trace, how much traffic is there with destination port 80 or how many nodes having an out-degree of more than 3000 and have transmitted more than 40MB data.

Some examples of simple usage:

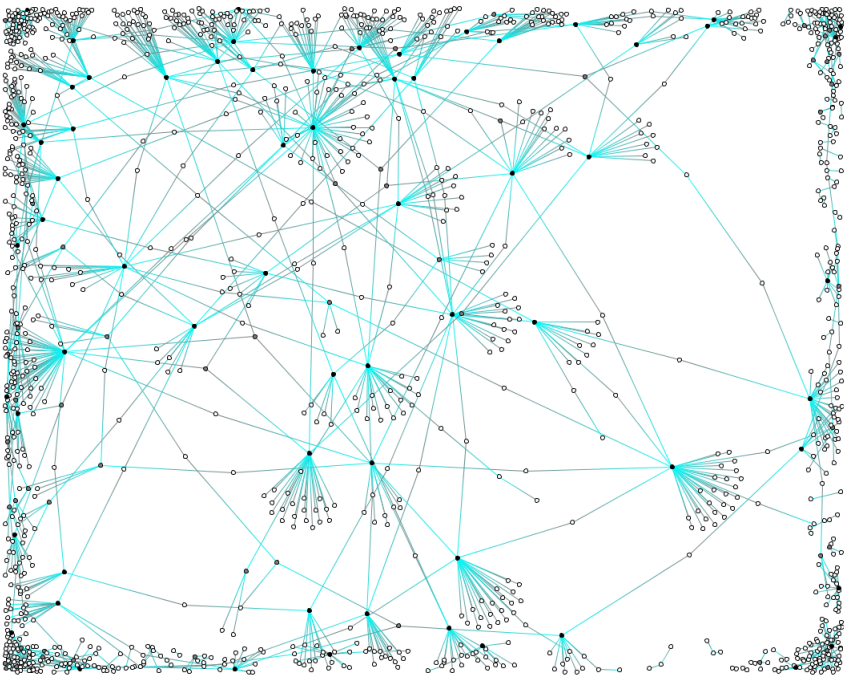


Figure 9: Searching for traffic originating from any port larger than 65530, here we see normal communication

5.1.2 Advanced

Advanced usage is to use the applications main feature, the visualization to draw conclusions and to do more queries from those conclusions. This is useful for example to find malicious activity, like bot nets, worms and

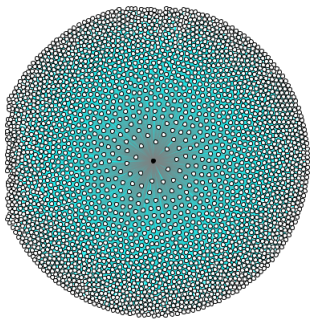


Figure 10: Searching for traffic on port 79 (finger), here we see a potential scanning attack

trojans, or new phenomenons like P2P systems trying to hide themselves by using well-known ports and/or tunnel itself via HTTP or something similar.

Data mining In this usage example, we are looking for interesting patterns in a random port interval. The port interval was selected randomly in the port range used for dynamic or private ports.

Filter used: source-port>64999 and source-port<65101

This gives us a graph with mix of unknown application traffic with a source port between 65000 and 65100, but we might be able to discover an interesting node that we would not be able to find otherwise.

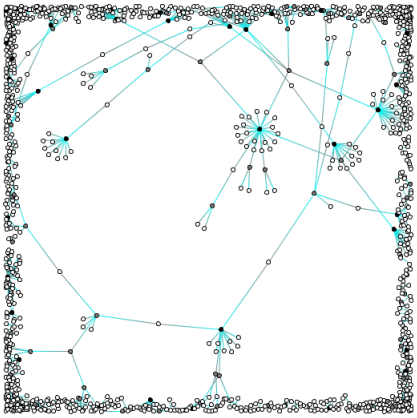


Figure 11: All flows with a source port between 65000-65100.

In the upper right corner there is a host that is connected to at least three hosts that have a large number of connections. Here is a closer look with the node marked with a square.

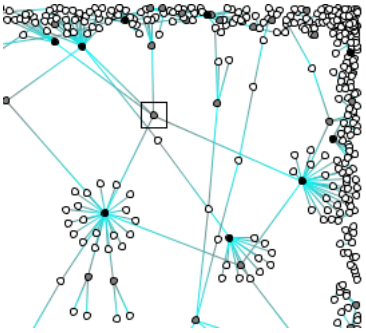


Figure 12: The square marks a node connected to several nodes with in turn has many connections.

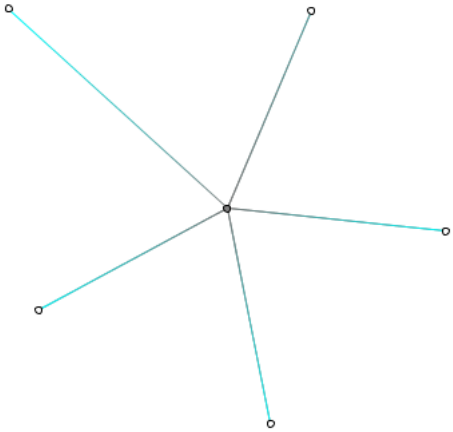


Figure 13: The node communicates with only a few other nodes.

A count of four nodes connected to the center node shows that they are communicating with several thousands other nodes in our 20 minute trace.

- Node 1: 13870 nodes, 26910 edges
- Node 2: 14475 nodes, 28287 edges
- Node 3: 11266 nodes, 22025 edges
- Node 4: 9751 nodes, 19058 edges

Since the number of edges is almost double the number of nodes, the communication is clearly going in both directions. From this we can conclude that the four nodes probably is not performing a scan. If the traffic had only been going from node 1-4, it could have been an indication of a botnet or similar.

6 Conclusions

In this thesis, we have presented a visualization tool for network traffic with filtering capabilities. This tool can be used to analyze any network trace that can be reduced to flows. The user can define filters to check for specific traffic patterns in large data volumes, for example port scanning. The application can also be used for data mining, which means starting out with traffic on a randomly selected port. Then explore it deeper by refining the filters based on the results of all the previous filters. It can also be used for discovering protocol patterns by analyzing traces from controlled environments, which can be useful when developing new protocols.

Our application is split into two parts: a backend responsible for data management and a front-end that is responsible for visualization and user interaction.

The backend is realized with a SQL database that is accessed by the front-end through standard SQL queries. The reason for using SQL queries is that regardless of whether we construct our own language or use an existing one, the user still has a learning curve. But learning something that is standardized and has a lot of available documentation should be easier and more meaningful.

The front-end is realized in Java because it is platform independent and has good third party graph libraries available. This is essential because the visualization is the core of our application and writing a graph visualization library from scratch is a thesis of its own.

The result is an application that is more of a proof of concept than a general tool and thus requires some expert knowledge to use. The user needs to be familiar with network concepts such as protocols and flows. The user also needs to know SQL syntax to construct the filter rules used for selecting data to be visualized.

That said the application is still very usable and since the target user group are network analysts, the knowledge needed to use the application is most likely covered.

7 Future work

This version of the application is a proof of concept. To be truly usable in a production environment a lot of improvements to the presentation of data must be made. Some examples:

- More information encoding, like edge thickness and drawing style
- Possible to change what affects thickness, like if it's amount of bytes or packets
- Extend the node information and make the information retrieval dynamic (fetch info when it is requested)
- Specialize the application for a specific kind of traffic, for example e-mail
- Add an exporter for GraphViz, because GraphViz can handle much larger graphs.

References

[1] Martin F. Arlitt and Carey Williamson. The extensive challenges of internet application measurement. *IEEE Network*, 21(3):41–46, 2007.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. `http://www.ietf.org/rfc/rfc1945.txt`, May 1996.

[3] Nevil Brownlee and K.C. Claffy. Internet measurement. *IEEE Internet Computing*, 8(5):30–33, 2004.

[4] Caida. `http://www.caida.org/`. [Online; accessed 25-July-2010].

[5] Douglas Comer and John C. Lin. Probing tcp implementations. In *USENIX Summer*, pages 245–255, 1994.

[6] Douglas E. Comer. *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture*. Prentice Hall, 5th edition, 2006.

[7] Coralreef. `http://www.caida.org/tools/measurement/coralreef/`. [Online; accessed 25-July-2010].

[8] Duncan Coutts, Don Stewart, and Roman Leshchinskiy. Rewriting haskell strings. In *Practical Aspects of Declarative Languages 8th International Symposium, PADL 2007*, pages 50–64. Springer-Verlag, January 2007.

[9] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286. ACM, 2006.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2068.txt`, January 1997. [Online; accessed 25-July-2010].

[11] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. *Lecture Notes in Computer Science*, 2259:1–??, 2001.

[12] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement, 1991.

[13] Graphviz. `http://www.graphviz.org/`, 2010. [Online; accessed 16-Mars-2010].

[14] Internet assigned numbers authority. `http://www.iana.org/`. [Online; accessed 25-July-2010].

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

[16] JGraph. `http://www.jgraph.com/`, 2010. [Online; accessed 22-Mars-2010].

[17] Wolfgang John and Sven Tafvelin. Heuristics to classify internet backbone traffic based on connection patterns. In *ICOIN '08: 22nd International Conference on Information Networking*, 2008.

[18] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[19] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM, 2004.

[20] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240. ACM, 2005.

[21] Hyun-Chul Kim, Kc Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and Kiyoung Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *ACM CoNEXT 2008*, 2008.

[22] V. P. Kumar, T. V. Lakshman, and D. Stiliadis. Beyond best effort: router architectures for the differentiated services of tomorrow's internet. *Communications Magazine, IEEE*, 36(5):152–164, 1998.

[23] MySQL 5.1 Reference Manual. 7.4.4. how mysql uses indexes. `http://dev.mysql.com/doc/refman/5.1/en/mysql-indexes.html`, 2009. [Online; accessed 22-October-2009].

[24] Anthony Mcgregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. *Passive and Active Network Measurement*, pages 205–214, 2004.

[25] J. Mignault, A. Gravey, and C. Rosenberg. A survey of straightforward statistical multiplexing models for atm networks. *Telecommunication Systems*, 5(1):177–208, 1996.

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

[27] M. Perenyi, D. Trang Dinh, A. Gefferth, and S. Molnar. Identification and analyis of peer-to-peer traffic. *Journal of communications*, 1(7):36–46, 2006.

[28] Skype. `http://www.skype.com/`, 2009. [Online; accessed 21-October-2009].

[29] Spotify. `http://www.spotify.com/`, 2009. [Online; accessed 3-November-2009].

[30] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit wep in less than 60 seconds. Cryptology ePrint Archive, Report 2007/120, Apr 2007.

[31] Voddler. `http://www.voddler.com/`, 2010. [Online; accessed 24-July-2010].

[32] Wikipedia. Caesar_cipher — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Caesar_cipher&oldid=318415052`, 2009. [Online; accessed 21-October-2009].

[33] Wikipedia. Bloom filter — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Bloom_filter&oldid=346894015`, 2010. [Online; accessed 24-July-2010].

[34] Wikipedia. Traffic flow (computer networking) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Traffic_flow_(computer_networking)&oldid=355521842`, 2010. [Online; accessed 24-July-2010].

[35] Carey Williamson. Internet traffic measurement. *IEEE Internet Computing*, 5(6):70–74, 2001.