DEGREE PROJECT IN INFORMATION TECHNOLOGY, SECOND LEVEL
STOCKHOLM, SWEDEN 2016

Graphical system visualization and flow display

A visual representation of an authentication, authorization, and accounting backend

JOAKIM AF SANDEBERG

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Graphical system visualization and flow display

A visual representation of an authentication, authorization, and accounting backend

Joakim af Sandeberg

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

28 July 2016

Examiner: Gerald Q. Maguire Jr.

Abstract

Displaying the architecture of a software system is not a simple task. Showing all of the available information will unnecessarily complicate the view, while showing too little might render the view unhelpful. Furthermore, showing the dynamics of the operation of such a system is even more challenging.

This thesis project describes the development of a graphical tool that can both display the configuration of an advanced authentication, authorization, and accounting (AAA) system and the messages passed between nodes in the system. The solution described uses force-based graph layouts coupled with adaptive filters as well as vector-based rendering to deliver a view of the status of the system. Force-based layout spreads out the nodes in an adaptive fashion. The adaptive filters starts by showing what is most often the most relevant information, but can be configured by the user. Finally, the vector based rendering offers unlimited zoom into the individual nodes in the graph in order to display additional detailed information.

Unified Modeling Language (UML) sequence charts are used to display the message flow inside the system (both between nodes and inside individual nodes).

To validate the results of this thesis project each iteration of the design was evaluated through meetings with the staff at Aptilo Networks. These meetings provided feedback on the direction the project was taking as well as provided input (such as ideas for features to implement).

The result of this thesis project shows a way to display the status of an AAA system with multiple properties displayed at the same time. It combines this with a view of the flow of messages and application of policies in the network via a dynamically generated UML sequence diagram. As a result human operators are able to see both the system's architecture and the dynamics of its operation using the same user interface. This integrated view should enable more effective management of the AAA system and facilitate responding to problems and attacks.

Sammanfattning

Att visualisera arkitekturen av ett mjukvarusystem är inte lätt. Visas all tillgänglig information så blir vyn för komplicerad medan ifall för lite visas så blir vyn onödig. Att samtidigt visa dynamiken som uppstår när systemet arbetar är ytterligare en utmaning.

Detta examensprojektet beskriver hur utvecklingen av ett grafiskt verktyg, som både kan visa konfigurationen av ett avancerat autentisering-, tillåtelse- och bokförings-system (AAA) och meddelanden som skickas mellan noder i systemet. Lösningen använder en kraftriktad graflayout tillsammans med adaptiva filter och vektorbaserad rendering för att visa en vy av systemets status. De adaptiva filtren börjar med att visa den information som oftast är mest relevant men kan ställas in av användaren. Nyttjandet av vektorbaserad grafik tillhandahåller obegränsade möjligheter för användaren att zooma in på delar av grafen för att visa mer detaljerad information.

UML sekvensdiagram används för att visa medelandeflödet inuti systemet (både mellan noder och inuti noder).

För att utvärdera resultatet av examensprojektet blev varje iteration av designen utvärderad vid möten med personalen på Aptilo Networks. Dessa möten gav återkoppling på vilken rikting projektet tog samt input med t. ex. idéer på nya egenskaper att lägga till.

Resultatet av detta examensarbete visar ett sätt att visa statusen för ett AAA system med många av systemets egenskaper visade samtidigt. Det kombinerar detta med en vy av flödet av meddelanden och applikationpolicies i nätverket via ett dynamiskt genererat UML sekvensdiagram. Resultatet av detta är att mänskliga operatörer kan se både systemets arkitektur och dynamiken i hur det fungerar i samma gränssnitt. Detta gränssnitt bör möjliggöra mer effektiv hantering av AAA systemet och underlätta lösningar på både problem i systemet och attacker mot systemet.

Acknowledgements

I would like to thank Aptilo Networks for providing the project opportunity for this thesis project and in particular Chris Steinbach for providing valuable insight into the ALE configuration system. I would also like to thank my examiner Gerald Q. Maguire Jr. for examining me and providing feedback on my work.

Contents

List of Figures

List of Source code listings

List of Acronyms and Abbreviations

This document requires readers to be familiar with some technical terms relevant to the thesis. Below is a list of these terms.

AAA	Tripel-A system. A networking system that handles Authentication, Authorization, and Accounting. This system keeps track and controls the users access to network resources.
Adapter	An interface to the ALE system which can receive and send information, either inside the system or to external networks via protocols such as RADIUS. One adapter can be linked to more than one node for redundancy.
ALE	Aptilo Longterm Evolution. A network backend and AAA system developed by Aptilo Networks.
Application Group	One or more nodes in the ALE system which together provides an application such as a database or history storage.
JSON	Javascript Object Notation. Data structure used to store virtual objects.
Node	This term refers either a vertex inside a graph or a machine in the ALE system, depending on context.
Protocol	Either a networking protocol or a way to evaluate information flowing to an adapter in the ALE system, depending on context.
Regular Expression	Regex. A search pattern used to match specific text inside a string of text. Support special characters such as ***[0-9]*** for numbers between 0 and 9, ***[a-zA-Z]*** for any lower or uppercase letter between a and z and **+** (plus

	sign) for one or more characters of a given match. As an example, the regex ***([a-zA-Z][0-9]+)*** will search for all strings containing a sequence with one or more letters followed by a number of any length, e.g. *test0312* but not *monkey* or *13214test*.
Ruleset	One or more rules associated with an Application Group. The rules in the ruleset determine how to process information inside the Application Group.
SVG	Scalable Vector Graphics. An image format which stores images as vectors instead of pixels.
Web GUI	Web Graphical User Interface. The Web GUI referenced in this report is the web interface through which the ALE system is monitored and configured.

Chapter 1

Introduction

This thesis is the result of a thesis project carried out at Aptilo Networks A graphical system overview has been added to their Wi-Fi and cellular backend service to make it easier to administer. The graphical system overview is designed to display relations in the network as well as information flows inside this network. Each flow displays which components of the system are communicating and what data is being sent between them. The system also displays in detail what happens to specific ”traced messages” inside the system. These details can be shown both for individual servers and between servers.

1.1 Background

This section will describe Aptilo Networks as a company, why large scale networks are needed and the characteristics of these large scale networks, in particular Aptilo Networks' technologies.

1.1.1 About Aptilo

Aptilo Networks[1] is a company headquartered in Stockholm, Sweden, that produces and markets software systems to manage mobile data and Wi-Fi services for 3G, LTE, WiMAX, Wi-Fi, and fixed broadband networks, including solutions for mobile data offloading using Wi-Fi. Aptilo's service management platform controls billing, along with user services and access to the network. The company offers service management and policy-based control solutions for both telephony network operators and Internet service providers(ISPs). In addition to its headquarters, Aptilo has regional offices in Kuala Lumpur, Malaysia; Dallas, Texas; and Dubai, United Arab Emirates[2].

1.1.2 Large scale networks

Large scale network systems are becoming more and more prevalent in today's society. For example, in the UK there is one Wi-Fi hotspot for every 11 people and worldwide there is one for every 150 people[3]. Many of these hotspots are connected to larger networks administered by ISPs and wide area cellar network operators. To administer these systems a backend is used to verify users, keep track of sessions, and connect the various types of hotspots to the network operator's network. These hotspots may support various protocols, differ in their behavior, and be spread over large geographical areas.

In order to manage large scale Wi-Fi systems, Aptilo Networks has created a platform called the Aptilo Service Management Platform™ (SMP)[4] which consists of user portals, billing system, and an authentication, authorization, and accounting (AAA) service. This service management platform is realized using the Aptilo Long term Evolution (ALE) architecture. This architecture has three layers: management, control, and execution. The management layer is used for configuration and monitoring. The control layer interacts with external systems to control these systems using protocols such as RADIUS, DIAMETER, BGP, SNMP, SOAP/XML, etc. The execution layer provides application logic processing.

According to Chris Steinbach of Aptilo[5] SMP system handles all the routing and control of the network. A typical SMP deployment consists of between one and several dozen SMP nodes as well as connections to external nodes (such as Wi-Fi access points or cellular basestations). Each of the SMP nodes handles one or more services, such as policy and service control, session storage, client traffic control, or database storage.

In addition, Chris Steinbach said that each of the SMP nodes may also have one or more adapters to handle communication between SMP nodes or with external nodes. Such adapters include an HTTP adapter, RADIUS adapter, and DHCP adapter. Each node also has several additional properties, such as which site it is at, what different protocols it can handle, and its current load. To get a full grasp of of the system with all the different connections and protocols interacting, a graphical representation is needed. This graphical representation can present different views depending on whether the user wants to see the structure of the system or to follow a sequence of messages due to a given interaction with a user.

1.2 Problem

The number of SMP nodes and external nodes (typically ranging between 5-50 SMP nodes, 10-200 external nodes) makes it infeasible to present all of the available information at once, thus some abstractions need to be made. The problem with such abstractions is that important information may be lost and some abstractions might make the status of the system harder to understand instead of easier.

The location of the nodes in the graph must be considered. The graph might have multiple desired properties that have to be weighed against each other and a balance between these properties has to be found. For example, it might be desired that nodes sharing a property (for instance their geographical location) should be positioned close to each other, but this might lead to more edges crossing inside the graph making the edges harder to follow. This means that a balance has to be found in which these nodes might be positioned close together but without voiding some other condition for instance minimizing the number of edges that cross.

Due to the large number of users and high amounts of traffic it might be hard to identify specific traffic and to monitor the flow of information between nodes inside the system. However, a desirable feature is that a network administrator can turn on a trace flag to identify the traffic of a specific user or protocol, e.g. RADIUS authenticates. This traffic can then be displayed to show how this traffic flows through the system, while highlighting where any errors occur along the way.

From this problem a problem statement arises in the form of:

How can a program best present a schematic high-level overview of the SMP system, combined with a (possibly dynamic) traffic overlay?

1.3 Purpose

The purpose of this thesis is to describe the development and functions of a tool which represents a high-level overview of a distributed system with many different properties and protocols for each node. Additionally, the tool can present dynamic data about traffic propagating through the network.

1.4 Goal

Aptilo has clearly stated a set of requirements about how the graphical overview system should function and look. These requirements can be seen in appendix A. Note that in this appendix an SMP instance implemented using the ALE architecture is referred to as an ALE system (this same convention will be followed in the rest of this thesis). As stated by Aptilo, the key requirements are:

- "The ALE management interface MUST provide a consolidated, schematic high-level overview system configuration including ALE nodes, and external nodes, node groups and networks."
- "Given that some ALE installations will be complex, involving many nodes, it MUST be possible to control the level of detail[...]"
- "It SHOULD be possible to generate diagrams exposing system behaviour from user trace [...]"

These requirements constitute the goal of this project.

The finished tool will be integrated as an addition to their ALE management web Graphical User Interface (GUI) where two new pages will be added. One with the flow view and one with the network overview. These two pages will be combined to one if it is deemed feasible.

1.5 Sustainability

The environmental impact of the work described in this thesis can be considered benign. A better system overview will lead to a system with a more even load over the system and easier identification of bottlenecks. With this increased information the administrator will be able to expand the system only at the points where it is actually needed. This leads to fewer servers that needs to be kept running and this means less power consumed. The economic impact of the tool should be positive for the company as this tool should help make their system more attractive to their customers. Moreover, the societal impact should be positive as the administrators will be able to better analyze problems with the system, hence more quickly resolve problems; therefore, providing better service to the end users. The increased use of mobile data offloading using Wi-Fi has a very positive benefit for the end users in terms of longer operating times for their devices and in some networks lower cost for the service. Moreover, mobile data offloading means that there is less emitted radio energy over a large area, benefiting society.

1.5.1 Ethics

Special caution has to be taken when choosing how to do the evaluation of the thesis project as well as how information about the ALE system is described. No trade secrets about the ALE system will be mentioned, but enough detail has to be described to ensure that the reader understands the solution to the problem and how it enhances the existing ALE product. Additionally, the evaluation has to be done in a general and objective manner and critically analyzed. In this way aspects of the tool that are relevant to others and to other problems can be presented in fair and balanced way, potentially leading to the adoption of these elements in other tools and systems.

1.6 Method

The method adopted in this thesis project is the design science method using iterative design, implementation, and evaluation of a prototype. Each prototype will be demonstrated to the product development team at Aptilo and their comments will be combined with an artificial neural network to form an opinion of the current design. The process is iterated to improve the prototype until it has fulfilled the project's goal. The information from the series of evaluations will be analyzed using an inductive approach to form a conclusion. A qualitative approach will be used during meetings with Aptilo before reaching the final result, and finally a quantitative assessment will be done through the use of an artificial neural network to evaluate the final result.

1.7 Delimitations

This thesis will be limited to only looking at a representation of the system in a graph form with nodes and edges. No other type of system representation will be analyzed. The graphical representation will only be for rather small systems with approximately 50 nodes instead of large systems containing thousands of nodes. This means that some available graph abstractions will not be applicable to the project since some abstractions are only usable for larger systems.

The thesis will only focus on systems where each node has many shared properties and to be displayed at the same time. In a traditional graph each pair of nodes only might be connected or not.

The flow view will be specific for the ALE system and will only show traces of packets starting with a specific trigger event within the system. No traces between

external nodes will be done since the external nodes' program code can not be modified by Aptilo to ensure correct logging of the trace.

1.8 Outline

Chapter 2 will give a theoretical background to this thesis project. The key technologies used will be described and the current state of the art will be presented. The beginning of the chapter describes AAA networks and the ALE system. In the middle of the chapter graph visualization techniques are described along with how to analyze if a graph is easily readable and a description of the UML sequence diagram. The end of the chapter goes on to describe the different programming libraries and frameworks used to implement the solution.

In Chapter 3 the method for the work conducted is described. The beginning of the chapter describes how the literature study was done and what related work were found. The working methods used, Kanban and Extreme Programming (XP), are then described. The chapter describe how the rest of the work was conducted, starting with the presentation of mockups, how the graph was generated and displayed, and finally how the messages flows in the system were identified and displayed.

Chapter 4 explains how the evaluation of the result was done. The first part of the chapter describes the program Neuroph [6] which was used to create an artificial neural network to evaluate screen images with. The second part describes how the meetings with the staff at Aptilo were used to evaluate the work.

Chapter 5 the presents the results. Images of the different views that have been generated are displayed together with descriptions of how the user may interact with them. First images of the system overview is displayed, which shows a graph of the system together with a legend. Second images of the UML sequence diagram is shown.

In Chapter 6 an analysis is done of the data gathered from the artificial neural network. A graph of the results generated from the network is presented and its values are discussed.

Chapter 7 gives a conclusion to the thesis project. First a summary is given and then which goals were completed are mentioned. Next a final statement from Chris Steinbach is displayed. Lastly a discussion of the thesis project is given and future work for the tool developed is suggested.

Chapter 2

Theoretical background

This chapter will describe all of the theory underlying this thesis. The chapter begins with a summary of large-scale Wi-Fi systems and networks. This will be followed by short summary of some of the protocols relevant to this thesis project. Next the ALE system is described. Section 2.4 summarizes some important properties of graphs, while Section 2.5 presents the concept of force based graphical layout. Section 2.6 describes some techniques to enhance graphical layouts, while Section 2.7 discusses how we can evaluate whether a graph is readable. Section 2.8 describes a UML sequence chart. Section 2.9 describes the Ruby on Rails programming environment and Section 2.10 describes Javascript – as these formed the basis for the implementation. The last Sections 2.11 and 2.12 describe the how software development at Aptilo is structured and the working models that were chosen for this thesis project.

Appendix B was used to provide good and bad examples of graphs for training of the artificial neural network. Details of artificial neural networks can be found in standard textbooks on the subject, such as Christopher M. Bishop's "Neural networks for pattern recognition"[7] and "Pattern Recognition and Machine Learning"[8] by the same author.

2.1 Large scale Wi-Fi systems and networks

Creating a large scale Wi-Fi system is a non-trivial task. The systems delivered by Aptilo may have tens of thousands of Wi-Fi access points or basestations. Additionally, the network administrator needs to manage the traffic between these nodes and the various services, as well as managing the individual services and the servers on which these services run. Furthermore, many different protocols need to be understood. Access control is provided by an authentication, authorization,

and accounting (AAA) system. A traditional AAA system can control access to resources in the network, enforce policies for user groups or individual users, log usage, and provide usage information necessary for billing for use of services. Aptilo's access control system does all of this and more as it must also be able to interact with the network operator's AAA system. In the context of this thesis, AAA is described as:

Authentication Authentication in AAA means keeping track of which user is which so that no one can impersonate another user. This is most often done based upon a user name and a password, but might also be done by other methods, such as subscriber identification module (SIM) cards or certificates.

Authorization Authorization means that when a user is accessing the network he or she only gets access to those network resources for which they have subscribed. This might be used to assign the user a specific amount of bandwidth and grant access to certain internet services, such as web pages, but not to other services (such as video streaming).

Accounting Accounting is used to measure and record the amount of network (including service) resources a user uses. This accounting might keep track of the amount of data a user transfers, the amount of time the user is connected to the network, and what kind of data the user transfers. The accounting records can subsequently be used to either bill users or limit their usage after a certain amount of usage. Additionally, accounting data is also analyzed to extract network trends and to display aggregated resource utilization, both of which are important for network and service capacity planning.

2.2 Aptilo Long term Evaluation (ALE) system

The ALE system provides a complete AAA solution with a focus on flexibility. The system will be described by dividing the system into protocols, application groups, and adapters.

2.2.1 Different types of protocols

As previously mentioned there are many different protocols used to realize the AAA system. Some of the protocols used in the ALE system are:

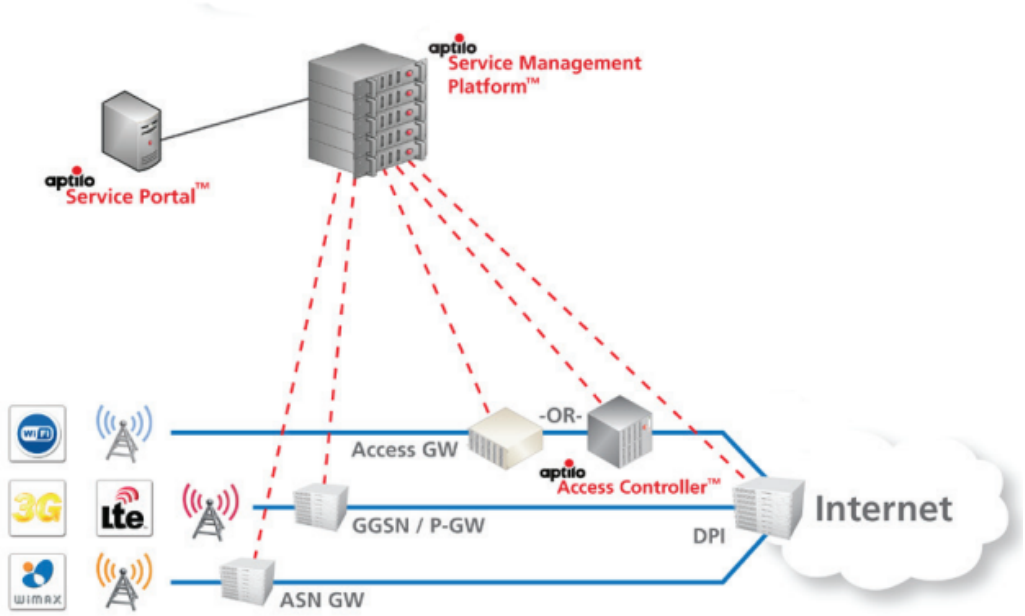


Figure 2.1: An overview of Aptilo SMP's interaction with the internet. The core of Aptilo SMP is the ALE system[1].

RADIUS and DIAMETER Both RADIUS and DIAMETER can be used to handle transactions for AAA. The Wi-Fi access points might make RADIUS or DIAMETER requests to the RADIUS/DIAMETER servers to determine whether to allow a device to access the network. These responses can specify the maximum bandwidth that this device can utilize. RADIUS is often used as an example throughout this thesis project but all these examples might be changed for another protocol for instance DIAMETER.

MAP/HLR The MAP protocol is used to communicate with the home location registrar (HLR) within GSM and UMTS mobile networks. This can be used to access the user's credentials when a mobile carrier is providing Wi-Fi access.

LDAP LDAP can be used to search for, retrieve, and

modify files on an external server. LDAP is frequently used to store data, such as logs for accounting purposes. LDAP can also be used to retrieve information about users in a distributed system.

2.2.2 Application groups

An application group in the ALE system is a group of nodes that together provide a service to the ALE system. Inside the application group both load balancing and redundancy are maintained by using the nodes within the group. The six types of application groups in the ALE system are:

Policy and service control Policy and service control is an application group dedicated to enforcing different kinds of rules depending on the traffic. Service control may for instance have a rule enabling users who have already signed in not to have to enter their password again.

Usage data The usage data application group keeps track of the usage of the system by users. It does this by keeping logs of the usage of each node and logging the events sent between nodes.

Storage cluster The storage cluster group stores long term information within the network. Such long term information includes user names and passwords.

History storage cluster The history storage cluster group has a similar function to the storage cluster group, but this group stores the user logs. These logs record which user is logged in, where, and for how long.

Client traffic controller The client traffic controller group maintains all the direct communication with nodes outside of the system and routes data from the user terminals (i.e., clients) to the internet and back. This group uses one or more adapters. When a user connects to an adapter the adapter applies policy and service control based upon the specific rules to be applied for this specific client.

Portal cluster A portal cluster provides a captive portal for users connecting to the network. This application group provides network specific web pages to the users.

Such a web page might display ”Welcome to network XXXX, please buy a voucher to surf”.

2.2.3 Adapters

Adapters within the ALE system are routines for handling different types of connections and network protocols. Currently there are seventeen adapters available, both the most common AAA protocols listed in Section 2.2.1, but also adapters for DNS, DHCP, SQL, and a custom API adapter. Most of these adapters can be customized; for example, to optimize the RADIUS adapter depending on whether it is communication with Aruba equipment or Cisco equipment. Another example of customization is to log usage data to an external database.

2.2.4 Tracing in the ALE system

An extra feature of the of the ALE system relevant to this report is the option to enable tracing. Tracing inside the system can be done by enabling a trace rule for a specific condition. The condition is a regular expression which is matched to either a standard field (such as username) or to any custom field setup in the policy for an adapter.

If the regular expression is true, then a message is placed in the log contain trace messages for all the following debug log messages that are generated. These trace messages give information about which trace was triggered and information about what is happening in the system, such as ”Evaluating policy rule-set [...]”.

2.3 Important properties for graph visualization

Important properties when visualizing a graph are hard to define and measure, since what makes a graph easily readable is to some extent subjective. However, general design aspects such as colors, fonts, line widths, and aspects specific for graph design, such as node clustering, line shapes, and node positioning may greatly affect whether a graph is easily to read or not. Unfortunately, what is a good property for a small graph might not be a good property for a large graph. For example when you have a large graph node edges might be better visualized if they are clustered together and then shown as wider links. However, for a small graph this clustering is unnecessary and makes the graph harder to read.

There exist some graphical layout solutions. For example, one widely used method to position nodes is to use force based layout. This method is described in the next section.

2.4 Force based graphical layout

Force based graphical layout is a way to determine positions for nodes in a graph using a physics based approach. The solution produces a graph where the number of edge crossings is minimized and nodes are positioned close to its neighbors.

Force based layouts positions the nodes by creating a n-body system where each node has a set of forces (gravity) pulling it towards other nodes. How strong the force is between two nodes depends on the number of links between these nodes. To ensure that all nodes are not pulled together spring like attraction/repulsion is used. This can be contrasted to the inverse square law applicable for a traditional gravitational system.

To describe the spring attraction and repulsion Hooke's law is used. Hooke's law describes how the force of a spring is proportional to the distance it is extended. This means that the springs in a force based layout will pull the nodes together. When they are closer than a certain threshold they instead will push away to ensure that the nodes in the system do not overlap. Different weights can be used for both the nodes and the edges in the system in order to arrange the nodes into a specific layout. A node with a higher weight will be harder to move by the forces acting upon it and an edge with higher weight will pull harder on the nodes it is connected to.

Certain steps can be taken to enhance the algorithm. Two highly useful things to add to the algorithm are slowing down when nodes are nearing their equilibrium and doing coarse prepositioning of the nodes. These are each described in the following subsections.

2.4.1 Slowdown when nodes are nearing their equilibrium

When nodes are nearing their optimal position they might reach an unstable state, such that the algorithm moves the node past its optimal position to a location on the other side of its optimal position. Then in the next iteration it moves it past the optimal position once again, This continues with the node never reaching its optimal position. To circumvent this problem the iterations can be made more fine grained at a heavy cost in performance, or the nodes can be slowed down when they are detected to be moving back and forth. Slowing down ensures that the nodes can move rapidly to the area near their optimal position, but when they are moving past this position they are slowed down by a factor, i.e. their effective weight is increased to make them stop moving.

2.4.2 Coarse pre-positioning of nodes

A pre-positioning algorithm can be applied to ensure that nodes start off close to their optimal positions. The pre-positioning algorithm used in this thesis was first presented by Hua, et al. in ”Force-Directed Graph Visualization with Pre-Positioning”[9]. Their paper describes how pre-positioning can be used to decrease the time to reach equilibrium by approximately 20%. This has the added benefit of making the result more consistent compared to a random initial positioning of the nodes.

The pre-positioning proposed can be described as:

1. V_{all} is all nodes in the graph. Sort V_{all} by node degree.
2. Select node V_k that is the node with highest degree in V_{all}. Position V_k in the middle of the frame of reference.
3. Find all nodes V_{kc} that are connected to V_k. Position all V_{kc} in an evenly spread out circle around V_k.
4. Find all nodes V_{kcc} that is connected to V_{kc}. Position all V_{kcc} in an evenly spread out circle around V_{kc}.
5. Repeat step 4 recursively for all children to V_{kcc}.

In each step only nodes that have not yet been positioned should be moved. Sorting by highest degree ensures that these nodes have the most effect on the layout.

Some enhancements to this algorithm can be made. The algorithm presented by Dong, et al. in ”An advanced pre-positioning method for the force-directed graph visualization based on pagerank algorithm”[10] takes a novel approach by using pagerank to determine the importance of nodes instead of using degree as Hua, et al. did. According to Dong, et al. this leads to a better layout than just using each node’s degree.

2.5 Other possible techniques to enhance readability

There exists many techniques to make a graph easier to read. This can be done by changing shape of the lines, filtering out unnecessary information, and changing how the information contained in the graph is presented. The following subsections describe several of these techniques, specifically edge clustering, filtering, and ways of showing shared properties.

2.5.1 Edge clustering

By grouping edges together the connections in the graph can be viewed in a more general setting[11]. This could be applicable to a graph where each edge represents one unit of traffic being sent between two nodes. When many edges exists between clusters within the graph, then these edges might be grouped together and displayed as one larger edge (as shown in Figure 2.2). The resulting graph would then be simpler to read and the edges represent the amount of traffic between regions instead of the amount of traffic between individual nodes.

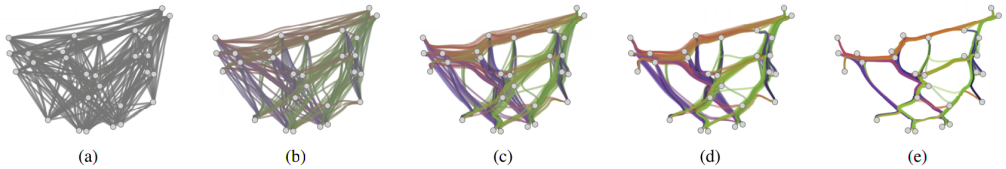


Figure 2.2: An animation sequence for an edge-clustering process. Color is used to encode edge directions[11].

2.5.2 Filtering

Filtering of the graph is a simple way to reduce the amount of information displayed to the user. Filters can be applied in many ways, such as filtering on domain specific properties or graph specific properties. Examples of domain specific properties are “only nodes with an RADIUS adapter” or “Hide nodes that are part of the storage cluster application group” while a graph specific property can be “Show nodes with a degree larger than 10”.

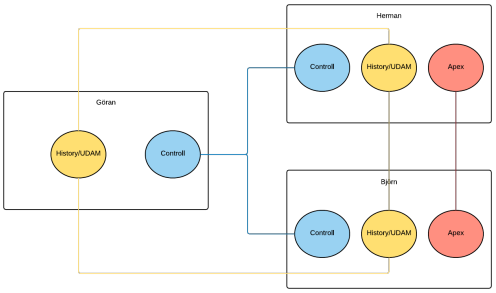
2.5.3 Different types of shared property representation

The edges in the graph might represent different properties shared by one or more nodes. For instance, edges might represent nodes that are part of the same subnet, part of the same application group, or simply nodes that have sent traffic between each other. This means that different ways to represent these edges might be used and combined to make the graph as easy to read as possible[12].

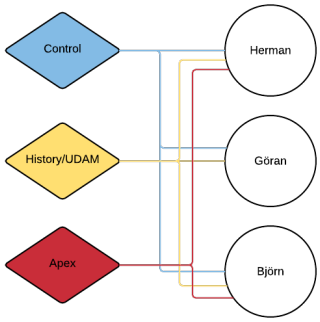
Complete graph

One way to show that a group of nodes share a property is to make the nodes a complete graph. This means adding an edge from every node in the group to every other node. This has the advantage of easily showing that the nodes are

connected and if the edges represent for instance a subnet it is intuitive to see that the connected nodes share a physical connection. The disadvantage of this approach is the large number of edges needed. The number of edges grows with $O(n^2)$ where n is the number of nodes. An example of such a complete graph is shown in Figure 2.3a.



(a) Properties represented by creating a complete graph with sub-nodes



(b) Properties represented by virtual nodes

Figure 2.3: Two kinds of system property representations. The two graphs present the same information but in two different ways.

Link to virtual nodes

To decrease the number of edges needed for a group, one approach is to add a virtual node to the group. This can for instance be done for a subnet by creating a node whose name is the name of the subnet and connecting all nodes that are part of this subnet to the new virtual node. This has the advantage of decreasing the number of edges needed from $O(n^2)$ to $O(n)$, but the result is a little less intuitive since there is an extra hop between connected nodes. An example of using virtual nodes is shown in Figure 2.3b.

Grouping

Instead of using links to display a shared property the nodes might instead be physically grouped together. This can be done by adding hidden edges between the nodes and letting the force based layout act upon these hidden edges to position the nodes closer together. For example, when combined with an edge filter, this can be used to group the nodes by physical location and display (color) edges depending on type of service. The resulting graph would then display which

services are distributed among multiple physical locations and which are not. An example of such an graph is shown in Figure 2.4.

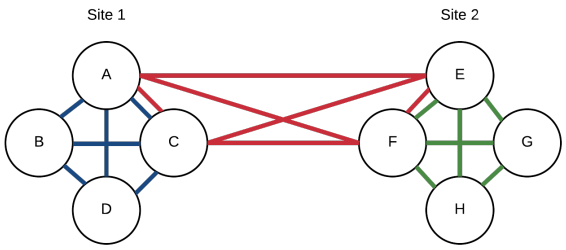


Figure 2.4: Displaying properties both as links and as grouping. The nodes are grouped by their respective site into site 1 and site 2. It is now clearly visible that the blue and green properties are each only present at one site each (site 1 and site 2, respectively) while the red feature is shared between the two sites.

2.5.4 Other approaches to represent shared properties

There are more approaches available than these, but most other approaches are either suited for much larger networks, or networks with rather different structure such as social networks. A few of these approaches are described in ”Motif Simplification: Improving Network Visualization Readability with Fan, Connector, and Clique Glyphs” by C. Dunne and B.Schneiderman. The three shapes Fan, Connector, and Clique Glyphs were considered, but ultimately were not used in this thesis.

2.6 Detect if a graph is easily readable

Relevant to this study is to evaluate if the resulting graph is good result in the sense that both the staff at Aptilo finds it useful and that it is good with regards to some objective measurement.

As described in Cody Dunne’s presentation [13], as mentioned on page 27, there exists more than 24 rules to take into account when measuring the readability of a graph. In Katherine Ognyanova’s presentation ”Static and dynamic network visualization with R” [14] four of the most basic measurements are:

- Minimal edge crossing,

- Uniform edge length,
- Minimal amount of nodes overlapping, and
- Symmetrical.

However, many other measurements are listed by various sources [15, 16, 12]. Some of these measurements are (in addition to the previous four):

- Minimize edge bends,
- Central placement of high degree nodes,
- Minimize difference in node size,
- Hierarchical layout (i.e. directed edges facing the same way as much as possible), and
- Avoid crossing among outlines (i.e. avoid having different clusters of nodes cross their outlines).

Unfortunately, no general tool to measure these qualities were found. The closest tool that was found is described in Cody Dunne and Ben Shneiderman's paper "Improving Graph Drawing Readability by Incorporating Readability Metrics: A Software Tool for Network Analysts" [17], but the program SocialAction described in the paper could not be downloaded. This is due to the fact that the SourceForge page [18] for the project had been abandoned. The page had not been updated since 2013 and contained no files. My examiner did find the code as part of the ManyNets project via the link `http://www.cs.umd.edu/hcil/manynets/manynets-source-code-final.zip`. I tried to build and run it both via IntelliJ, Netbeans, and CMD but kept encountering errors. As a result I did not use this software.

Instead a simple image recognition neural network was constructed to objectively determine if the graphs generated in the thesis are most similar to the graphs with many of these good aesthetics or similar to graphs with many bad aesthetics. The neural network is very general.

Neuroph [6] was used to construct and run this neural network. The image recognition network in Neuroph is constructed from the description available in the Neuroph image recognition guide. Two sets of images are given as input to the neural network. One with images the network should recognize, in this thesis these are graphs with good aesthetics. The other set is a set of images considered poor and should not be recognized, in this thesis these are graphs with bad aesthetics.

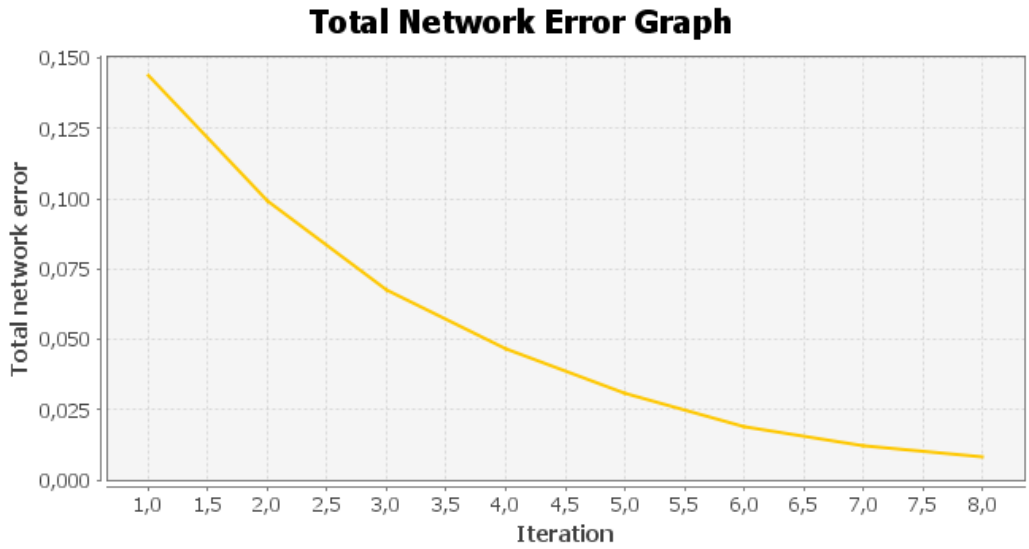


Figure 2.5: Graph showing the error of the neural network during learning. The graph is generated by default by Neuroph when training a network. A full iteration is considered when all images in the learning set has been passed once. This means that 1/30th of an iteration is the same as the network analyzing one image, when 30 images are used in the training set. The setup used when generating this graph is presented in Section 5.2 on page 58.

Before the neural network learns from the input images some parameters have to be set. The most important parameters for this neural network are

Image sampling size The images are split before passed into the neural network, into subimages of a given height and width. The number of neurons in the first layer is then H*W where H and W are height and width of the grid.

Hidden layers size Inside the neural network there are multiple hidden layers. More hidden layers and more neurons per layer will generally produce a more accurate result for the network, but adding more neurons give diminishing returns and takes more processing power on the computer system running the network.

Max error How large an error the neural network can have on the training set. This is the value to be reached by the error rate after multiple iterations in the graph (see for example the graph shown in Figure 2.5). Typically around 1%(0,01) maximum error is acceptable (in this

graph corresponding to 8 iterations), but this depend on the application. This is the value represented on vertical axis on the graph in Figure 2.5. A lower value means better accuracy for the network, but requires a longer training period and very low values are not guarantied to ever be reached when learning.

After training the neural network with the two input sets new images can be passed to the neural network. The neural network outputs a value between 0% and 100% indicating how well the images match those among the training images. This value will be used as a metric to objectively determine if the layout done by the program constructed in this thesis project can be considered good and to ensure that the work done during the thesis is progressing in the correct direction.

2.7 UML Sequence Chart

Since the project's goal is to both visualize the system as a whole and to visualize flows in the system, a UML sequence chart is used to visualize the flow independently of the structure of the system. The UML sequence chart is a well known way of displaying how a series of messages or actions are sent between multiple entities or actors within a system. In such a chart each actor is displayed with a timeline going vertically through the chart. Each event is then annotated by an arrow going between two actors or in a loop back to the same actor. Each arrow initiating or requesting something is solid, while arrows returning or ending something are dashed. An example of a sequence chart for ordering a waffle in a cafeteria is shown in Figure 2.6.

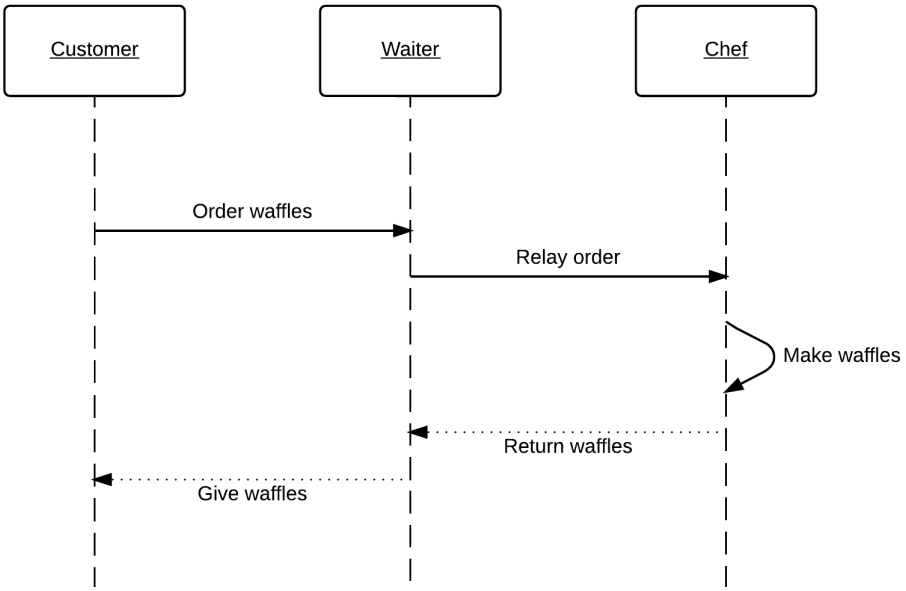


Figure 2.6: Example of ordering of a waffle as UML sequence chart.

2.8 Ruby on Rails

Ruby is a programming language created in 1995. Ruby is considered a functional programming language balanced with imperative programming. The main concept of Ruby is that everything can be considered an object and unlike languages such as Java where primitive types (classless types) exists, in Ruby even basic things such as integers (int) extends the class Numeric. A common application for Ruby is to host webpages via the Rails framework.

Ruby on Rails (or simpy Rails) is a web-application framework specially designed to include special tools and APIs to enable database-backed web applications in Ruby. Rails consists of three layers which each have a specific responsibility. Each of these layers is explained in the following subsections.

2.8.1 View layer

The view layer determines the layout and design of the webpage. Each view that should be presented to the user of the webpage has a layout specified in the view layer. The view files are often HTML files with embedded ruby calls. The embedded calls may execute calls to the model layer and thus get unique content for the current user.

2.8.2 Controller layer

The controller layer handles requests arriving to the Rails application. It decides which view should be returned combined with which model. The controller ensures that when a user requests a certain domain the correct view and correct model are paired and returned. This means that multiple views may share the same model and multiple models may be combined into a single view. For example, the controller layer may keep track of whether a user is logged in or not and either display a login prompt or the actual view that should be displayed if the user is logged in. The controller may keep track of the type of user and thus either display an administration panel or a regular user panel.

2.8.3 Model layer

The model contains the logic specific for the application. The model layer contains all the logic and functions related to the data associated with the site. The model layer retrieves/saves data from/to a database, and gives the view layer content to show. The model layer may also summarize and modify the data and execute different functions depending on what interactions the user has with view layer.

Active Record

Active Record is the persistent data management used in a Rails application as part of the model layer. Active Record provides an interface with Ruby functions to a relational database and provides storage of Ruby objects directly mapped to the database. To enable Active Record for an object the object is created as a subclass of ActiveRecord::Base. This creates a table in the database with the same name as the class and the object will have a number of Active Record API functions attributed to it. To fill the table with associations to other models tags, functions such as *belongs to*, *has one*, and *has many* can be used.

Active Record provides several functions to retrieve objects from the database. If the object User is stored in the database, then `User.all` will return all users stored in the database. `User.find_by(name: David)` will find the first user with the name ”David”. A more complex call to retrieve users can be `users = User.where(name: 'David', occupation: 'Painter').order(created at: :desc)`. This will return all users with the name David and occupation Painter and sort them by the date they were created in the system.

To get the same information as the previous Active Record statement in SQL the

query would be `SELECT * FROM Users WHERE name = David AND occupation=Painter ORDER By created at DESC;` The similarity to SQL in the function names are intentional, as Active Record does an SQL query to the database when it is used. The advantage of Active Record is that it provides Ruby functions and it creates Ruby objects from the return of the SQL query. Active Record also sets up the objects within the database and automatically adds their properties, just by sub classing an active record when creating an object.

When Active Record is used all changes made to an object after it is retrieved are kept in a buffer and **not** added to the database *until* the `save` function is used on the object. In the case of ALE, this means that when a user is making changes to the system no changes are actually saved until a specific save button is used to commit these changes to the database.

2.9 Javascript

Javascript is a loosely typed, lightweight programming language initially created by Oracle. Javascript is best known as a scripting language for web pages, but it can also be run outside of a web browser through frameworks such as Node.js. Javascript is typically run at the client side of the web page and enables the developer of the website options to manipulate and create live interactions with the website. Such interactions can be simple things (such as changing the page layout, detecting mouse movement, and enabling buttons) to more complex things (such as browser games, advanced graphical demonstrations and physics simulations). Some of the Javascript libraries that will be used in this thesis project are described in the following subsections.

2.9.1 Sigma.js

Sigma.js is a graph drawing library constructed in Javascript to display graphs on web pages. It can display graphs using one of three different renders: HTML5 Canvas, WebGL or SVG. This library also supports a variety of plugins.

Sigma.js's core functionality only displays nodes and edges given positions. Sigma.js can be extended to do many things through the use of plugins, e. g. show custom images for each node, display multiple edges between the same two nodes, create a layout using a basic force based layout, and parse graphs from external Javascript Object Notation (JSON) files.

The basic graph object in Sigma.js only has 10 methods that can be used to retrieve one or all edges/nodes, add edges/nodes, and remove edges/nodes. Most

of sigma's functionality comes from its easily extendable core as well as its high rendering performance.

The following paragraphs describe the main plugins used in this thesis

sigma.renderers.customShapes

This plugin enables custom node shapes as well as overlaying images on top of nodes in the sigma graph. This means that system nodes and external nodes can have different shapes. It also means that images with node properties can be overlaid in order to display the value of a node's attributes.

sigma.exporters.svg

The sigma.exporters.svg plugin enables export of a Sigma.js graph to a Scalable Vector Graphics (SVG) file. This SVG file can then be combined with custom shapes to enable drawing graphs within nodes, where the internal node structure can be displayed with a graph. This graph can then be overlaid onto the node and displayed as a part of a larger graph. An example of this concept is shown in Figure 2.7

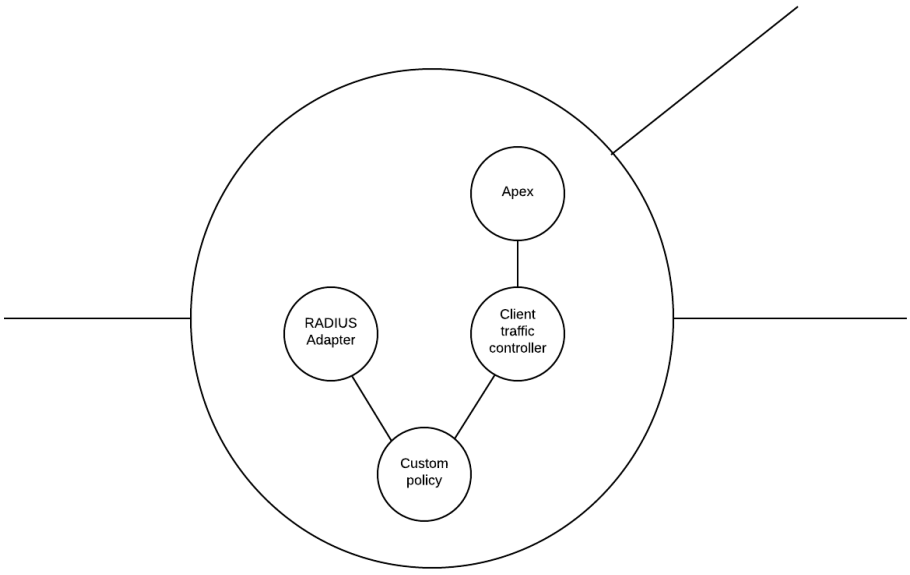


Figure 2.7: An example of a node's structure overlaid onto a node to create a graph within a node.

sigma.layout.forceAtlas2

The sigma.layout.forceAtlas2 plugin gives sigma the functionality to do force based layout of the sigma graph. The plugin takes several options as input, such as node and edge weight factor, starting iterations, and how much to slow down the nodes should be subject to. How the force based layout works was further described in the Section 2.4 starting on page 12.

sigma.plugins.filter

The sigma.plugins.fliter plugin enables the use of custom functions to specify filters for the graph. The filter function is called once for each edge and node in the graph and returns either true or false. If true, then the currently called node or edge is displayed, otherwise it is hidden. The filter function can consider properties such as node type, node degree, edge, length, or a custom property to decide if the element should be shown or not.

sigma.renderers.parallelEdges

By default Sigma.js does not support multiple edges between two nodes, but the sigma.renderers.parallelEdges plugin adds this support. Each edge in the graph has a count property and to have multiple edges between the two nodes different counts are set for the different edges. All edges between two nodes can then be displayed at the same time without overlapping.

2.9.2 cardinal-spline-js

Cardinal-spline-js[19] is a project available on GitHub which adds a new function to the HTML5 Canvas. The new function receives an array of points as input and outputs a shaped containing a curved line through the points. The curve is interpolated via the formula for a cardinal spline. An exapmle of the shape before and after the cardinal-spline-js function is used can be seen in Figure 2.8.

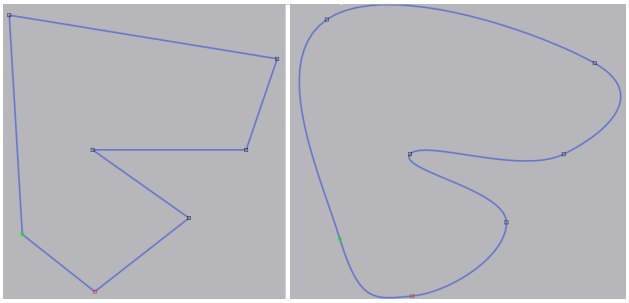


Figure 2.8: A line drawn through a set of points. The image (to the left) shows the straight lines between points before cardinal-spline-js is used and the resulting curve (to the right) after cardinal-spline-js has been used.

2.9.3 D3.js

D3.js[20] is a Javascript library made for processing and representing data on web pages. D3 stands for Data Driven Documents. Like Sigma.js D3.js supports plugins. In this thesis project D3.js is combined with a plugin to present UML sequence diagrams. The plugin is d3-message-sequence.

The GitHub project d3-message-sequence enables the drawing of sequence diagrams with D3.js. This plugin has functions for adding messages between actors as well as parameters for setting animation properties when adding data. During this thesis project this plugin was used without any animations but the plugin was modified to support different kinds of arrows, extra labels for the arrows, and different arrow colors. These modifications are further described in Section 3.8 starting on page 44.

2.10 Interface to ALE system's data

The ALE system configuration is accessed through a web page served via Ruby on Rails. This means that a developer might access the data either from the SQL database where Active Record stores its values or via the built-in Rails functions that Active Record provides. The various configuration objects such as adapters, system nodes, protocols, rulesets, external nodes, etc. are represented and made accessible via these two interfaces.

Active Record can be used to retrieve the configuration of the system, but to retrieve data recorded by the system the log file is accessed. The ALE system stores data in a log file aggregated from all nodes in the system locally on the master node. The master node also runs the web server from which the system

is configurated. This logfile can be accessed directly from Ruby code running on Rails, but has to be parsed into usable objects. This parsing is described in Subsection 3.7.3 starting on page 43.

2.11 Development at Aptilo and Bugzilla

The development at Aptilo uses Bugzilla[21] as its main form of organization. Bugzilla is used by many software projects (i.e. the Linux Kernel, Open Office, Red Hat and by Mozilla). While Bugzilla was originally only intended to track bugs in the system, Aptilo has extended its usage to include new features to implement and other changes to be made to the system. Each new feature to implement is added as a new ”bug” and all relevant fields, such as name, who it is assigned to, priority, severity, and target release are filled out.

Bugzilla is well integrated with Git and together with each bug the relevant Git repositories gets listed. When a developer has made changes to a file, the developer may send the changes for code inspection. The code inspection part of Bugzilla has been modified at Aptilo to work through Bash Unix Shell via the command *codeinspect*. The command accepts multiple arguments but the main arguments are

Bugnumber The tracking number in Bugzilla, so the reviewer knows which bug the changes are made for,

Reviewer email The mail to which the *codeinspect* is sent,

Git branch The branch in which the new commits to be inspected are located, and

Commit hash If the developer does not wish to send the full git history since the last accepted commit, the hash of a commit may be specified.

The commits sent in the *codeinspect* may only add one feature at a time, and should contain two commits with each *codeinspect*; one with the modified files, and one which increases the version number.

After a *codeinspect* has been sent, the reviewer receives an email containing a git patch with the suggested commits. The reviewer might then forward the message if need be, to people in the staff with more expertise in the changed code. The reviewer then responds back to the sender, either with changes needed or with a confirmation that the developer can push the changes to the remote server.

2.12 Working model

Kanban[22] is an agile method to use for software development and extreme programming(XP)[23] is a programming philosophy used when writing program code. Both of these working models were originally designed to be used in a team of developers, but many parts are applicable for a sole developer.

2.12.1 Kanban

Kanban[22] was first developed at Toyota in the 1940s. The motivation was to implement just in time development in a similar fashion to how a convenience store manages its stock. This means that each step in the development process tries to start new work only when it deems all previous work is finished, and to have as few jobs running in parallel at a time. This is supposed to lead to more flexible planning and faster output.

In Kanban a priority board lists of all the tasks in the backlog. The board at Aptilo is grouped by Later, Soon, and Next. An example of the planning board can be seen in Figure 2.9. The most important task or tasks are positioned in Next, while in Soon a small group of slightly less important tasks are placed. Finally, in Later are all tasks which are currently not deemed important. When a task is finished, then a new task is plucked from the Next group. When the Next group is empty, then the most important tasks in the Soon group are moved up to the Next group. This means that only the most important tasks are being worked on and no lengthy planning sessions are needed. This approach can be compared to Scrum where each sprint is planned beforehand.

Figure 2.9: Picture of the priority board during this project

2.12.2 Extreme programming

Extreme programming (XP)[23] is a philosophy with an emphasis on simplicity. This simplicity is meant to be used as the simplest approach most often is the best approach when programming. XP also emphasize the importance of integrating code with the excising solutions to avoid compatibility issues. XP also suggests a structure of the work that goes very well with Kanban in that the most important piece work is selected and finished before starting on any new work. A flowchart of how XP might work is illustrated in Figure 2.10.

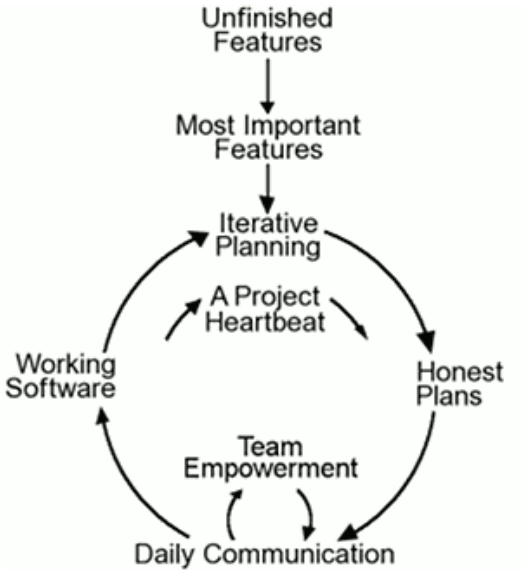


Figure 2.10: Flowchart for a typical XP approach

These two working models(Kanban and XP) were chosen since Kanban was used as a standard at Aptilo, I had previous experience with XP, and I believed they would go nicely together.

Scrum was considered as another option but were not chosen since Kanban was a better option to use, due to Kanban being standard at Aptilo.

Chapter 3

Method

The overall goal of this thesis project is to demonstrate a practical solution which is capable of displaying the state of an ALE system with many different properties visualized along with how the traffic is flowing within the system. This chapter will describe the method used to achieve this overall goal

3.1 Literature study

The first part of the project was a literature study. The study's goal was to get a general understanding of different types of graph representations and to find existing solutions to similar problems.

The literature study is based upon sources found by querying Google Scholar, IEEEXplore, and Scopus. The search queries were formed with a basic understanding from previous courses at KTH, specifically: *ID2220: Advanced Topics in Distributed Systems*, and *IK1550: Internetworking*. These searches were done in several iterations based on the previous search results in order to get more and more specific results. At the end of the literature study a reading list of approximately 15 documents were used as a base. From this knowledge the first prototype was created to demonstrate to the staff at Aptilo. Their feedback was the main input for my subsequent design decisions.

3.2 Related work

Six previous works were deemed to be the most relevant to this thesis project and highly related to the work to be conducted. A description of each of these works as well as which parts of the work were relevant and why is given in this section.

Force-Directed Graph Visualization with Pre-Positioning

Hua, et al. presented a model for prepositioning the nodes in the graph before applying a force-based layout in ”Force-Directed Graph Visualization with Pre-Positioning: Improving Convergence Time and Quality of Layout”[9]. This was highly useful for the thesis project since it described a rather simple algorithm for pre-positioning the nodes which decreases the time to reach equilibrium by approximately 20%. Another benefit of pre-positioning (instead of spreading the nodes randomly) is that the result will always be the same after a certain number of iterations with the force-based algorithm. This means that a user of the system will always end up with the same graph for the same input parameters, compared to having the nodes in a random pattern when starting the algorithm. This is especially useful for the user when sharing the generated graph's settings with a colleague, since they both will see the same result.

There exists more advanced pre-positioning algorithm such as one based on pagerank described by Dong, et al. in ”An advanced pre-positioning method for the force-directed graph visualization based on pagerank algorithm” [10], but since the graph constructed by the tool in this thesis project was rather small, the performance benefit of such advanced pre-positioning was not needed. Using this advanced pre-positioning would only increase the complexity.

Improving Graph Readability by Incorporating Readability Metrics

In [17] by C. Dunne and B. Shneiderman several *readability metrics* are defined for how to determine if a graph is easily readable. These *readability metrics* include properties such as node occlusion, edge crossing, edge crossing angle, and edge tunneling. All these *readability metrics* are always kept in consideration during this thesis project to ensure that the graphs presented and generated by the tool always follow these guidelines in the best possible way. A tool called SocialAction[18] was created, but as described in Section 2.6 on page 16 the source code for the tool could not be found early in this project.

Measuring and Improving the Readability of Network Visualizations

In the Ph.D. thesis ”Measuring and Improving the Readability of Network Visualizations”[16] by Dunne there are further descriptions of how a graph can be determined to be easily readable or not. The contents of this thesis overlap with the work presented by Dunne and Shneiderman in ”Improving Graph Drawing Readability by Incorporating Readability Metrics: A Software Tool for Network Analysts” (described in the previous paragraph). Since Dunne's thesis describes many other useful things to be considered in this thesis project such as ”Group-in-a-Box” layouts, and describe the *readability metrics* with other words, this was considered a key source.

Exploiting aesthetic criteria

In ”An improved force-directed graph layout algorithm based on aesthetic criteria”[15] by Dong, et al. a force-based layout algorithm is presented which takes the aesthetics of the graph into consideration. The algorithm's result is influenced by some of the aesthetic criteria previously presented by C. Dunne, such as number of edge crossings and crossing angles. After the layout is done by the force-based algorithm a refinement process is described which curves the edges to improve the angular resolution yielding better crossing angles for the edges. Both the new layout and the bending of edges are highly relevant to this thesis project.

Comparing Expected and Real-Time Spotify Service Topology

In Vilius Visockas' master's thesis ”Comparing Expected and Real-Time Service Topologies”[24] a way of finding, visualizing, and analyzing the topology of Spotify's music service is described. The work described in the thesis is similar to this thesis project since the work described also visualizes a network's topology and flows. The major difference is that the scale of Spotify's music service is much larger than the typical ALE system, but many parts of Visockas' work were still applicable to this thesis project.

Exploring Color in Interface Design

In the article ”Exploring Color in Interface Design”[25] by Shubin, et al. describes how different colors can interact in an interface. The article describes how the properties of colors in the interface such as contrast, hue, and saturation can play a part in how a user sees various elements in the interface. The examples given in the article are used as guidelines when deciding what colors should be used in the tool created in this thesis project.

3.3 Presenting mockups and prototypes

Each Friday the development staff at Aptilo has a demonstration session at which the developers may demonstrate new features that they have added to the system and then lead discussions about problems and possible choices for their current work. During this project the current state of the prototype was demonstrated every other Friday and questions about how to proceed were asked. For the first Friday some mockup pictures were presented to give an idea of what the first prototype might look like. These demonstrations were also used to ensure that the work proceeded in the correct direction, so that in each iteration the graph either gave more information or became easier to read.

3.4 Retrieving information

To retrieve information SQL queries were first made directly to the database where Active Record store its information. This seemed like a good approach at first approach since I already had some expertise in SQL. A first implementation was constructed using SQL and Python to create a data structure for a basic graph. The disadvantage of this approach was that the SQL database does not contain any *uncommitted* changes to the configuration. These changes are only stored in the current state of Rails' ActiveRecord and thus I needed to use the ActiveRecord API to retrieve this information. To gain access to this uncommitted information the program and subsequent development were therefore changed to be completely in Ruby on Rails.

Retrieving the needed information within Ruby on Rails were rather easy with the #all accessor which retrieves all Objects of a given class. The start of the program therefore contained lines such as:

```
@system_nodes = SystemNode.all
@external_nodes = ExternalNode.all
@client_networks = ClientNetwork.all
```

3.4.1 Parsing the retrieved elements

Once the information had been retrieved it was parsed into useful objects. The different groupings of the nodes had to be created from this information, e. g. each application group had to be added to the graph as links between all nodes of the same type. Connections between system and external nodes were added as a virtual node with multiple links from each external node and one link to each system node. Other properties such as sites and subnets were also added to realize a complete graph. The last step was to export the collected information as a temporary JSON string. This JSON string is used as input to the graphical Javascript component.

3.5 Creating abstractions

The graphical component receives the graph as a very dense graph with a very large number of edges between some pairs of nodes. It is not uncommon for a pair of system nodes to share four subnets, three application groups, and a site. This information is given to the graphical component as the nodes having eight edges between each other, each marked with the corresponding edge type. To ease the reading of this file, filters are applied to the different properties so that a

user might choose to display only sites, only applications, or a combination some of different properties. The whole range of properties are shown in Figure 5.3 in Section 5.1. The figure is available on page 49.

3.6 Displaying the information

Sigma.js is used to display the information contained in the graph. To better display the nodes Sigma.js was extended in several ways. Some of these extensions (e.g. edgeDots - see Subsection 3.6.3) were found to be general enough to be included in the Sigma.js GitHub project. These plugins can therefore be found in the plugins folder in Sigma.js GitHub repository* or among the current pull requests.

Some of the source code is also shown in this thesis. The version of this source code that is shown has been changed to simply show function calls (rather than the full code and arguments) to make it clearer for the reader. For instance the existing code to add a line to the canvas might simply be displayed as an `addLine()` function.

3.6.1 Node properties displayed within nodes

To make it easier to view the properties of a single node within the graph, Sigma.js was extended with the customShapes plugin. This plugin was initially intended to display images within a graph so it had to be modified with a new custom shape for the purpose of overlaying a custom made graph on the image. The graph to be overlaid was generated with the exporter.SVG plugin. This combination produces the result shown in Figure 3.1.

* Sigma.js plugins `https://github.com/jacomyal/sigma.js/tree/master/plugins`

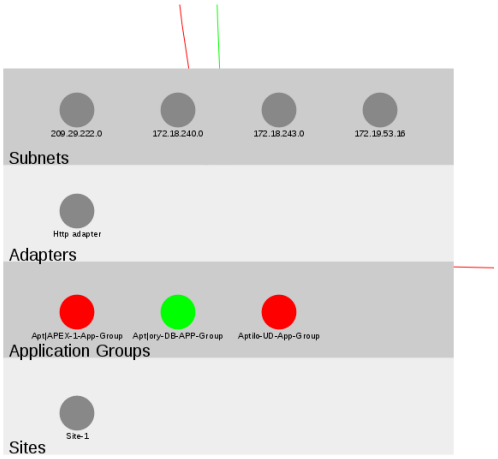


Figure 3.1: A node within the graph with its properties displayed.

3.6.2 Common properties as areas instead of links

If nodes only could have one property of a certain type (for instance a node can only be at one site at a time) it would better to display this property as an area instead of as a complete graph of the properties. To make this possible Sigma.js was extended with another customShape that keeps track of the edge of the shape spanning all included nodes. To increase the performance corners of the area not inside the view port are not rendered. To make the area rounded instead of a polygon with sharp corners the utility library cardinal-spline-js was used. An abbreviated version of the code to do this is displayed below.

Source code listing 3.1: Code to display an area behind a set of nodes in Sigma.js

```
1   /* Nodes are stored in the map in sub-arrays based on their
       group name (offset-name). Each node is then stored in the
        sub-array as[x, y, id]*/
2   var shapeToNodesMap=[];
3   var drawCustomBackground = function(node,x,y,size,context) {
4       /* Order the nodes by  their angle compared to the
           center, with angles moving in a clockwise direction
           starting with the node with lowest ratio between dX
           and dY. This is used to draw the outline of the area
           in the correct order. If this where not done the
           outlining would just jump between nodes randomly*/
5           var degreeMap = [];
6       var nodes = shapeToNodesMap[node.shapeName];
7       {minX, minY, maxX, maxY} = getMinMaxCoords();
8       var middleX = (maxX + minX)/2;
9       var middleY = (maxY + minY)/2;
10      nodes.forEach(function (node) {
```

```
11              var dX = node[0]-middleX;
12              var dY = node[1]-middleY;
13              var angle = Math.atan(dX/dY);
14              degreeMap[angle].push(node);
15          });
16          degreeMap.sortByKey();
17          var nodesToDraw = getNodesToDraw(degreeMap);
18          /* Add the nodes to an array and remove nodes that are
               not part of the edges of the area. This removal is
               done if the node is inside the triangle created by
               the middle point of all nodes in the group and the
               location of its two neighbours in the degreeMap. All
               nodes are of the form [x, y]. This means [0][1] is
               the first node's y-coordinate*/
19          context.moveTo(nodesToDraw[0][0], nodesToDraw[0][1]);
20          var pointsToCurve = [];
21          for (i = 0; i<nodesToDraw.length; i++){
22              var previousIndex = (i-1)%nodesToDraw.length;
23              if(previousIndex < 0){
24                  previousIndex = nodesToDraw.length+previousIndex
                        ;
25              }
26              var previousNode = nodesToDraw[previousIndex];
27              var nextNode = nodesToDraw[(i+1)%nodesToDraw.length
                    ];
28              var currentNode = nodesToDraw[i];
29              if(!isInsideTriangle(middleX, middleY, previousNode
                    [0], previousNode[1], nextNode[0], nextNode[1],
                    currentNode[0], currentNode[1])){
30                  pointsToCurve.push(currentNode[0]);
31                  pointsToCurve.push(currentNode[1]);
32              }
33          }
34          /* Create a nice looking curve, fill it and stroke it.
               The curve function is from cardinal-spline-js*/
35          context.curve(pointsToCurve, 0.5, 25, true);
36          context.stroke();
37          context.fill();
38      };
```

The above code starts off by retrieving all the nodes belonging to the current region and finding the middle coordinate of the region. It then proceeds to map the angle of each node relative to this middle coordinate and sorts the nodes by this angle.

To finally create the shape the points are looped through and if the node is considered to be an edge node (not inside the triangle created by the previous node, next node, and the middle) it is added to the shape. The shape is then filled

and stroked. The stroke creates some margin between the nodes and the edge of the shape. The resulting shape will now show the area behind the nodes inputed to the fuction.

3.6.3 Link status as dots upon the links

To show the status of the nodes, dots were added to the links between nodes to show if the link has both connected nodes online or offline. If any of the two connected nodes are offline, then the link is marked with red dots and if both is online the link is marked with green dots. An example of this is shown in Figure 3.2.

Figure 3.2: An example of a graph with colored dots at the end of the edges.

If the status is unknown for a node, then the dots are removed. The plugin made for Sigma.js is in the form of a new edge renderer by the name dotCurve. The code is displayed below in an abbreviated form.

Source code listing 3.2: Code to display colored dots on the links between nodes in Sigma.js

```
1 sigma.canvas.edges.dotCurve = function(edge, source, target,
    context, settings) {
2       addLine(); /* The previous edge renderer which adds the
          line between source and target*/
3       /*Check if there are any dots to be rendered*/
4       if(edge.sourceDotColor != undefined || edge.
          targetDotColor != undefined) {
5         /*Retrive the size and offset. If they are undefined
              default to 3 and 1 */
6         var dotOffset = edge.dotOffset || 3;
7         var dotSize = edge.dotSize || 1;
8         /*Scale the sizes to the current zoom level, defined
              by size and sSize*/
9         dotSize = size*dotSize;
```

```
10              dotOffset = dotOffset*sSize;
11              /*Create dots for the source and the target (if they
                    are defined)*/
12              if(edge.sourceDotColor != undefined) {
13                  createDot(context, sX, sY, cp, tX, tY, dotOffset
                        , dotSize, edge.sourceDotColor);
14              }
15              if (edge.targetDotColor != undefined){
16                  createDot(context, tX, tY, cp, sX, sY, dotOffset
                        , dotSize, edge.targetDotColor);
17              }
18          }
19      };
20      /* Function which adds a dot to the context depending on an
           offset, a size and a color*/
21      function createDot(context, sX, sY, cp, tX, tY, offset, size
            , color) {
22          context.beginPath();
23          context.fillStyle = color;
24          /*Get the coordinates for the point*/
25          var dot = getPointOnBezier(sX, sY, cp.x, cp.y, tX, tY,
26              offset);
27          /*Create a 2*Pi radians arc, i. e. a full circle*/
28          context.arc(dot.x, dot.y, size * 3, 0, 2 * Math.PI,
29              false);
30          /*Fill the circle with the previous fillStyle*/
31          context.fill();
32      }
```

In this code sX, sY, tX, tY are the source and target's X/Y coordinates (respectively). cp is a control point to determine the bending of the Bezier curve. dotOffset is the distance between the node and the dot on the edge and dotSize is the size of the dot.

3.6.4 Active legend

A feature that greatly improved the readability of the graph was the option to click the elements in the graph to display more information about them. For instance a user might click on a link to see its connections and what the link represents. To enable this functionality an "Active legend" was created next to the display of the graph. This active legend receives events when a user clicks on elements in the graph and depending upon the element the relevant information is shown. As an example, when the user clicks a system node the available information for the system node is shown, such as adapters, application groups, subnets and name. If the user clicks on an edge in the graph, then the active legend shows the start and end of the edge, what type of information it represents (subnet, application group

etc.), and which specific relation it represents (e.g. a specific subnet or specific application group). Images displaying this active legend is available in Figure 5.4a and 5.4b on page 53.

3.6.5 Layout engine

The layout engine is based upon the sigma.layout.forceAtlas2 plugin (in the following text simply forceAtlas) for Sigma.js, but this plugin was extended to support different filters for the layout. An example of the result of this modified algorithm can be seen in Figure 3.3. The nodes were initially spread out in a circle sorted randomly, but were then positioned by the algorithm based upon their color. The green nodes were filtered out when the positioning was done, thus they remain in a circular pattern.

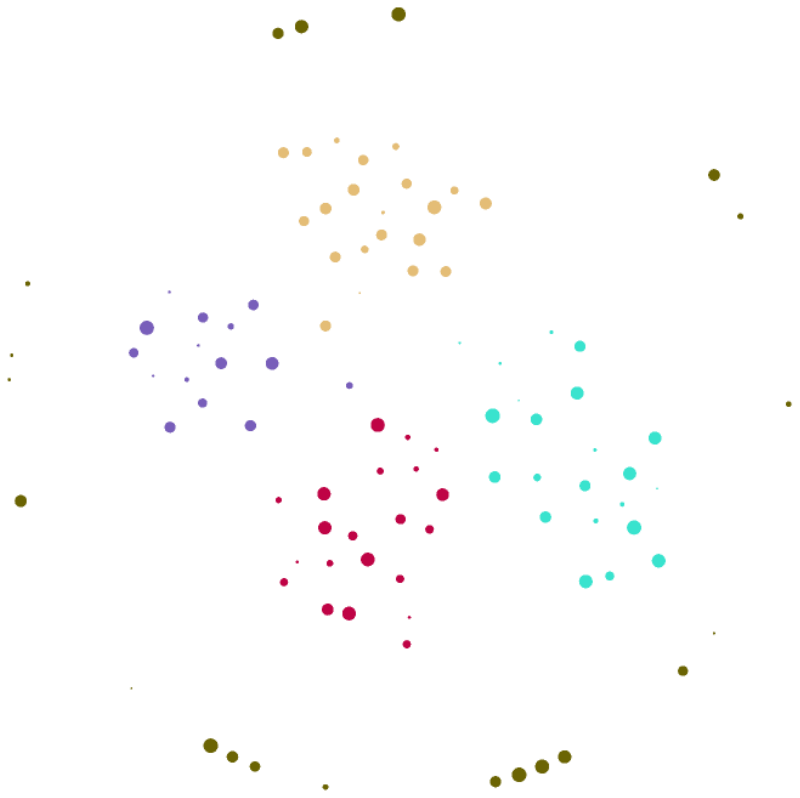


Figure 3.3: A layout created with the modified version of forceAtlas2 algorithm.

The new version of the algorithm receives two new optional properties as input. The two arrays `edgeFilter` and `nodeFilter` contain Strings. The Strings filters all the nodes and edges in the graph (respectively). The edges and

nodes in the graph have a new, optional, property `filterType` which is what the filters map against. If the filterType is present in the respective array, then the forceLayout will be applied to this element. Otherwise the elements are ignored when applying the forceAtlas algorithm. The modification can also be found in Sigma.js pull request 726*.

3.7 Generating flow information

This section describes how the flow information was retrieved and displayed. First a description of how a flow is identified is given, then how this flow is retrieved from the logs, and lastly how the flow is displayed in the web GUI.

3.7.1 UUID-tracking

A universally unique identifier (UUID) is generated by the Linux UUID library call (see the Linux manual page uuid(3)). This function generates a unique 16 byte long identifier. This identifier is used throughout the system. When printed to the log the last four characters from the (see the Linux manual page uuid_parse(3)) function is used. Four is an arbitrary number (set in the source code with the constant UUID_LENGTH) and the number of digits could be increased if lots of information is being logged (hence ensuring that the UUIDs do not overlap). The reason to not log all 16 bytes is for the log to be easier to read by humans. A long 16 byte tag would take up lots of space for the reader of the log.

To add this UUID-tracking several of the ALE systems packages had to be modified, including:

evh2 The event handler in the system.

aptlog The system's logging service. This package writes logs for the full system on the current master node.

freeradius The RADIUS component in the system receives, parses, and sends RADIUS requests.

sessionproxy-server The proxy relays requests to the correct node in the network for instance to a RADIUS adapter when receiving a RADIUS request.

* Sigma.js pull request 726 https://github.com/jacomyal/sigma.js/pull/726

3.7.2 Tracking counter

The UUID added to the tracking messages shows which flow the current flow is part of, but does not show the order in which the events have happened. Events at a given node will be ordered in the log file, but events between nodes will not be ordered. The clocks of each node were set using NTP and their accuracy measured using the `ntpq` command. The time stamps in the system were considered unreliable since the clocks in the system were measured to differ by about 70 ms. Also checking the order in the log file is not be reliable since the messages are not certain to be written in the order that they were sent over the wire. An example of this was that a request is sometimes logged after their corresponding ruleset was logged as having been evaluated. There are other was to improve this measurement to be better than 70ms, for instance by updating the NTP time more often or switching protocol to a more accurate one (such as PTP). Unfortunately, since the accuracy needed were likely to be higher than the accuracy NTP has to offer, and switching to PTP seemed harder than an alternative approaches, none of these approaches were pursued.

To solve this problem a counter was introduced to the message. The counter was increased at key parts in the code, for instance when the message was passed through evh2[26] and through freeradius. This means that a message with a higher counter will always have been created after a message with a lower counter. Unfortunately, this is not true if the message ”splits” into two parallel events. Since the ALE system is based upon distinct events, and the execution inside one of these distinct event is considered sequential in ALE, this is not a problem[5]. If the event were to split up, the highest counter is used by each node before increasing it, leading to a mechanism similar to lamport time[27]. If we were to track multiple ALE events, a more advanced kind of ordering would have to be used.

The policy engine in the system was also modified to more thoroughly log what it does by adding a log line before a policy is executed and when it has finished. From these log lines another sequence chart can be made which shows the calling of different policies within the policy engine. An example of how the log entries might look when transformed into a sequence diagram is visible in Figure 3.4. The log entries has been abbreviated to only show a relative time stamp and the log message.

1:Begin Aptilo-Timer-Schedule-RS
1:End Begin Aptilo-Timer-Schedule-RS
3:Begin Aptilo-CDR-RS
3:Begin Aptilo-WiFi-CDR-RS
4:Begin Aptilo-WiFi-CDR-Send-Event-RS
10:End Aptilo-WiFi-CDR-Send-Event-RS
11:End Aptilo-WiFi-CDR-RS
11:End Aptilo-CDR-RS

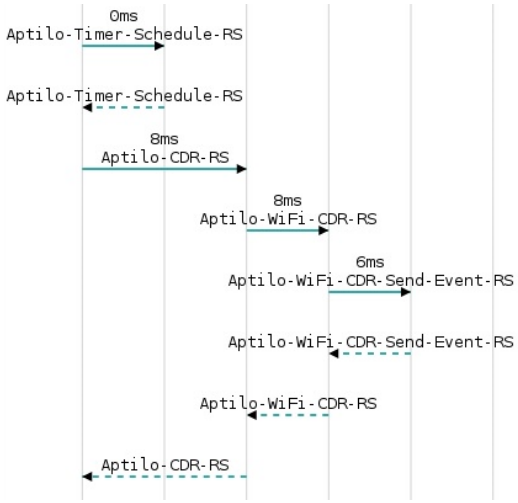


Figure 3.4: An example of the sequence diagram for a group of log entries. The log is displayed to the left and the corresponding sequence diagram to the right. The log only shows the relevant messages, and their relative timestamp.

3.7.3 Parsing the log

After the log was modified to include the UUID and the counter it needs to be parsed to extract each single UUID as an event sequence. A sample log message which had matched the tracking condition would now look like:

```
20160415 09:18:48.129 2 sessionproxy/SessionProxy[3193]: Trace:
    (User-Trace): [1]: "joakims-trace": e2a3|4: Sending RADIUS-
    Auth to 192.168.120.221:16066
```

The parts of the message could be viewed as:

```
Date Timestamp NodeId EngineType/EngineName[PID]: Type: (
    LogLevel): [TraceID]: "TraceName": UUID|Count: Message
```

Splitting the log message into an array at the character ' ' and String ": " gives each component of the log as an individual String. The indexes of the arrays are:

- For the String splitted at ' ' -
  - Index 0 - Date
  - Index 1 - Timestamp
  - Index 2 - NodeId
  - Index 3 - Engine

- For the String splitted at ”: ” -
  - Index 1 - Type
  - Index 2 - (LogLevel)
  - Index 3 - [TraceId]
  - Index 4 - ”TraceName”
  - Index 5 - UUID|Count
  - Index 6 and higher - Message

To ensure that the message to be parsed actually is a correct trace message the arrays are checked to have the correct length. Index 5 of the array derived from ”: ” which should contain UUID|Count, hence it is checked to match the regular expression [a-f|\d]{’+UUID_LENGTH’} which ensures that the text actually matches the correct format. UUID_LENGTH is a constant set to the number of characters of the UUID in the log. If the expression does not match the log line is discarded.

All of these properties of each log line are parsed into Ruby objects which are then grouped by UUID. The UUIDs are then sorted based upon the first time stamp for each UUID and internally on the counter for the UUID. To detect where to add messages in the sequence chart each time both the engine has changed and the counter has increased the previous log line is shown as the line that caused the message.

When dealing with the policy engine a “calling stack” is created so whenever a new policy is executed it is added to the stack and when a policy is done executing it is removed from the stack. This ensures that is always possible to know between which two policies the calls are interacting.

3.8 Displaying flow as a sequence chart

Displaying the flow as sequence chart is done with D3.js[20]. As noted earlier D3.js is a framework for displaying data driven documents on web pages. D3 provides functions for displaying and updating information on a HTML Canvas or SVG. As described earlier in Section 2.9.3, the GitHub project d3-message-sequence provides a simple plugin to display some of the properties of a sequence chart. These components were used to display the data in the web GUI. However, the plugin was modified to handle some special features, specifically:

Update data in the chart The default d3-message-sequence plugin could not update the data in the graph with new information. To correct this a

`remove()` and an `instant_add()` function were introduced. These functions provides the functionality needed to update data by removing old data and adding the new data.

Different color for different arrows To color the arrows in different colors depending upon which UUID they belong to the plugin was extended to check for a color property in the messages. This property is used to color the arrows.

Show extra information above arrows To show the UUID and how long the flow for the specific UUID lasts, an extra text field was added above the message arrows. For the first message for each UUID this text field was filled with this UUID tag and how many milliseconds it was between the first and last message for the flow of the UUID. This time indicates how long it took for the system to process the full flow caused by the message.

Add click functionality to the arrows in the chart To easier get further details of the events in the chart the functionality to click on the arrows in the chart were added. The search engine for the log available on the website accepts multiple different arguments but the relevant argument were a regex matching anywhere on the full log line.

To give the user the option to click on the arrows a regex matching the UUID and range for the function call was generated, since the UUID is unique for each trace. It is also ensured by Aptilo's convention that the ':' character were used to separate the different parts of the log line. This means that the regular expression `(([A-F]|[0-9]){4})\|\d*:` will only match the UUID and its counter in the log. The `\d` character in the regular expression were changed to a regular expression for the range of the UUID's counter up until the next arrow in the sequence diagram. Finally a link leading to the search engine for the log with the generated regular expression as an argument were created and attached to each arrow in the sequence diagram.

Chapter 4

Result evaluation

This chapter describes how the results of the tool developed in this thesis project were evaluated. The two main ways of conducting the evaluation were the use of a neural image network to get an objective evaluation as well as meetings at Aptilo to get a subjective evaluation. How the evaluation with Aptilo were done was described in Section 3.3 starting on page 33.

As described in Chapter 2.6 a neural network is used to determine the quality of the result. As training for the neural network 30 images were used that had either been mentioned as an example of a good graph or an example of a bad graph in sources listed in Appendix B. It is worth noting that 30 images is a very low number of samples to train a neural network with but the network will still output some metric determining the images similarity. All images used for learning were mentioned as either *good* or *bad* by the authors, often in pairs with one *good* and one *bad* version of the same graph. The reason for only having 30 images during training was since there was very hard to find examples of good and bad graphs from credible sources. During both my literature study and an extra search specifically for training images, these 30 images were all that could be found.

The configuration used when evaluating the generated graphs used 1600 neurons in the first layer, 12 neurons in the second layer, 8 neurons in the third layer whose outputs were combined into a value. A sample of the configuration of the network can be viewed in Figure 4.1. The setup was done based upon the guide provided at Neuroph's webpage.

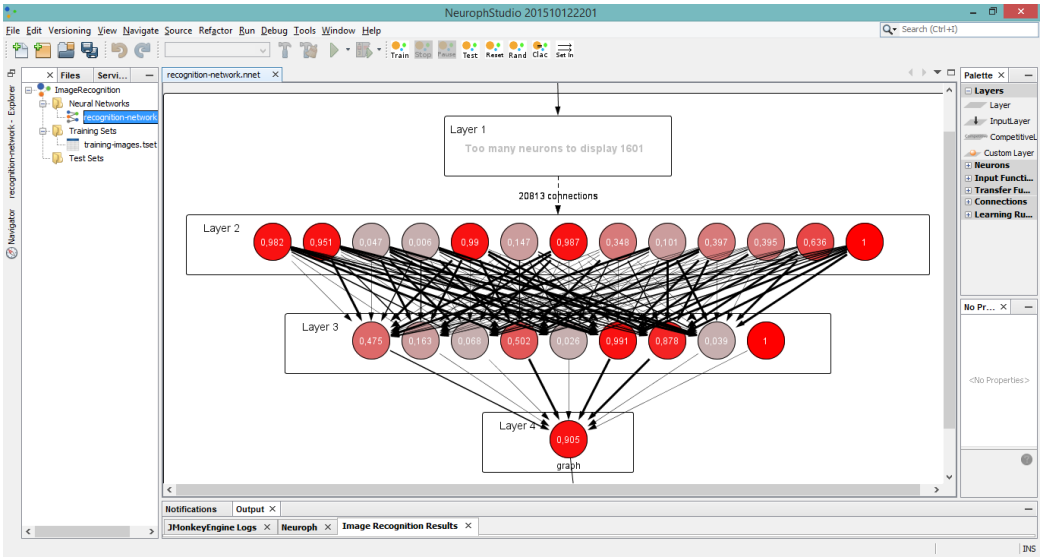


Figure 4.1: A graphical representation of the neural network. The values and color of the neurons show how much they each influence the final result.

To train the network the input were given for different number of iterations until the network showed the correct result with at least 99% accuracy, i.e., 1% error. This value was reached after 8 full iterations of the training set (meaning all training images has been inputted 8 times). The error during training can be seen in Figure 4.2.

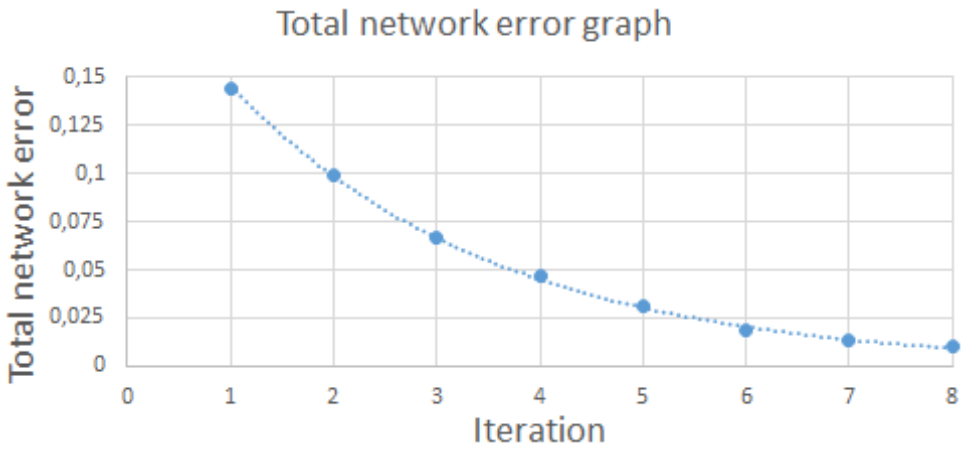


Figure 4.2: A graph representing the error when training the neural network. The error decreases as the network adjusts itself to the input. An error tolerance of 1% is used. The exponential trend line to the data has the equation $y = 0,2164e^{-0,393x}$ *and* $R^2 = 0,997$

.

Chapter 5

Presentation of results

This chapter presents the results collected during the thesis project. First images of the system's architecture will be shown followed by images of some example sequence charts.

5.1 Images of the graphs shown inside the web GUI

This section shows examples of the results as shown to the user as a page inside the web GUI.

Figure 5.1 shows a webpage giving an overview of the system. In the middle of the view is a graph which represents the system. To the right is an legend which changes when the user clicks on elements in the graph. At the bottom, the upper part of the settings panel for the graph visible. The full settings menu (accessible by scrolling down the page) is visible in Figure 5.3 on page 52. The menu items for the other parts of the web GUI have been redacted at Aptilo's request.

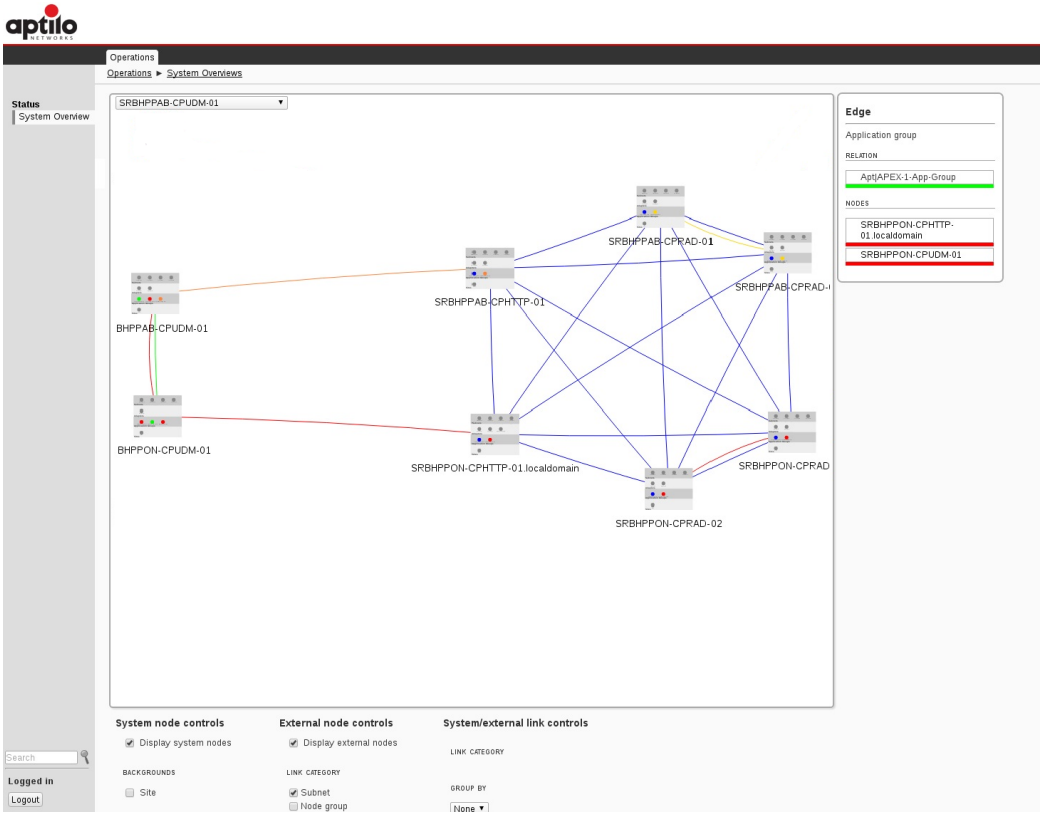


Figure 5.1: A view of how the webpage might look inside the Aptilo web GUI.

The images presented from this point forward will not show the rest of the web GUI, but instead only show the view presented inside the subpage. Since the user only accesses this part of the GUI from the same page, therefore the elements in the menu will not change. This means that the menu always will look the same on all images rendering the display of it redundant. Figure 5.2 shows another view

of the graph. The external nodes in the system can be viewed on the left side of the image as green dots. Labels for these are visible if the user hovers their mouse over one of these nodes. The ALE system is visible to the right. From the graph of the ALE system we can see that the system is spread out over two different sites (the two gray areas) with four nodes at each site. Additionally, there are two application groups spread over the two sites (visible as the blue group and the purple link).The structure of the two sites are the same since the upper half of the graph is a mirror image of the bottom half of the graph.

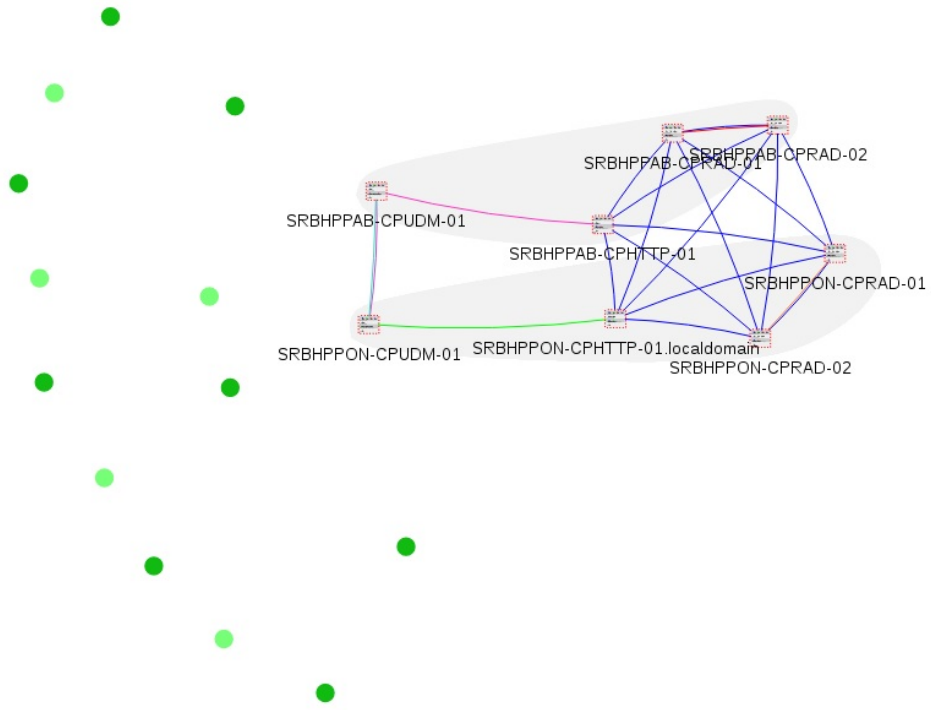


Figure 5.2: A view of the graph grouped by application group and site. The external nodes in the system are visible as green dots. The system nodes are the squares with each application group forming a complete graph.

Figure 5.3 shows the settings panel for the graph view. The settings panel is placed directly under the graph. When the user makes a change to the settings the graph is updated (nearly) instantaneously. If the layout is changed (accessed via the ”GROUP BY” drop down menu) an animation is visible when the layout is calculated so that the user can see how the grouping is done.

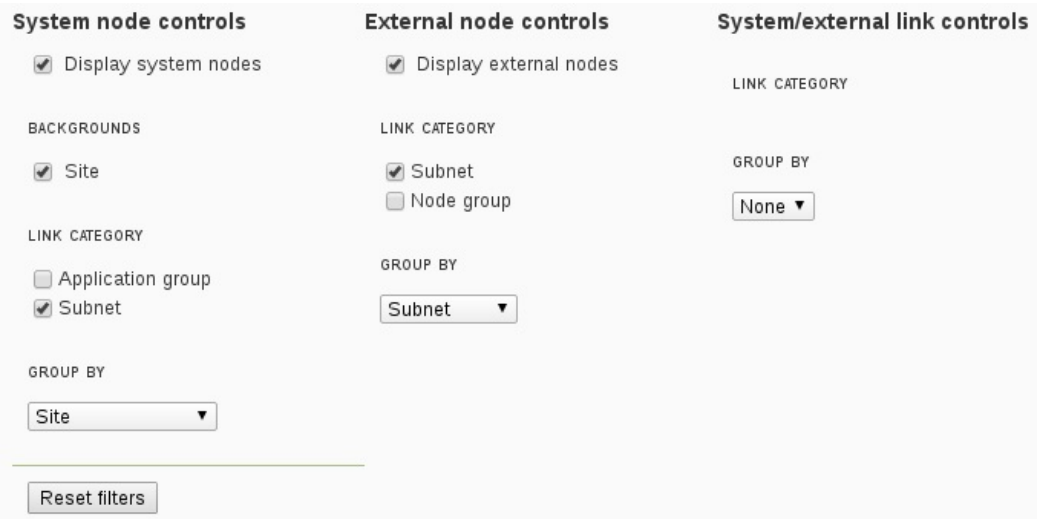
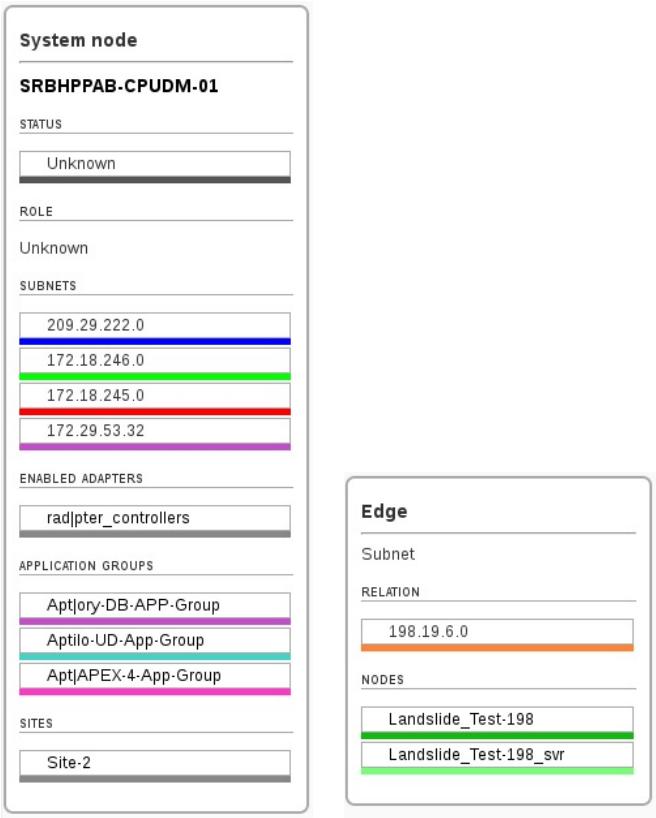


Figure 5.3: The settings menu below the system graph

Figure 5.4a shows how the legend looks when a node has been clicked. The available properties for the node which can be presented in the graph is shown. All elements in the legend are clickable to access the configuration page for the clicked property.

Figure 5.4b shows how the legend to the right of the graph looks when an edge has been clicked. It shows what relationship the edge represents, the type of the edge, and what nodes the edge connects.

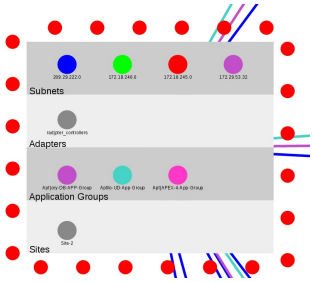


(a) Active legend displaying properties of a node

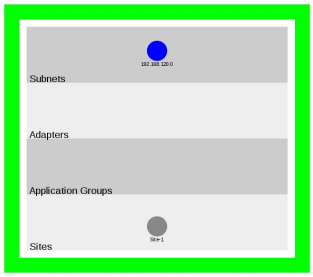
(b) Active legend displaying properties of an edge

Figure 5.4: Images of the settings menu which is displayed below the graph and the active legend which is displayed to the right of the graph.

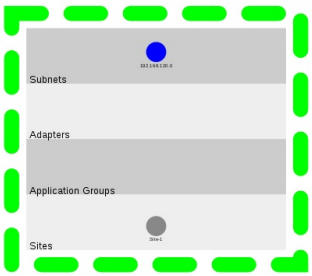
Figure 5.5 shows the different types of ALE nodes which are present in the system. Their types are either master, slave, or unknown combined with the node's status: online, offline, or undefined. Online nodes are nodes where the ALE system is up and running, while offline are ALE nodes which have been online at some point. The undefined nodes are systems which are starting, shuting down, or nodes which has not been connected yet.



(a) Example of a system node with its properties displayed inside it. The edge displays the current node status. Red is for offline and dotted is for an unknown type.



(b) View of an online master node. This by the solid green outline. Solid indicates master node, while green indicates online.



(c) View of an online slave node. The dashed green outline denotes an online slave node, where dashed indicates slave node and green indicates that the node is online.

Figure 5.5: Images displaying the different node types as well as the different edge representations. If the user zooms in on a node, they can see all of the properties of that particular node. This information is also available in the active legend if the user clicks on the node.

5.1.1 Examples of the sequence diagrams

This section shows examples of the sequence diagrams generated by the tool. These sequence diagrams are present on a separate webpage available via the system's menus. The menus visible in the screenshots in this report has been redacted to display only the current page. After the first figure the menus have been cropped out of the picture.

Figure 5.6 shows a sequence of messages between different components in the system. When the UUID changes from one UUID to another, then the color of the arrows changes to reflect this. Above the first arrow for the new UUID the time for the sequence to finish and the characters representing the UUID are shown. All arrows in the chart are clickable to access the log file at the entry which generated the selected arrow. The UUIDs shown in the chart are selected by the checkboxes above the graph. This means that the user might read the log, find an interesting UUID (for example each UUID at the time of an error) and display the flow at the time of the error by checking the box for the specific UUID.

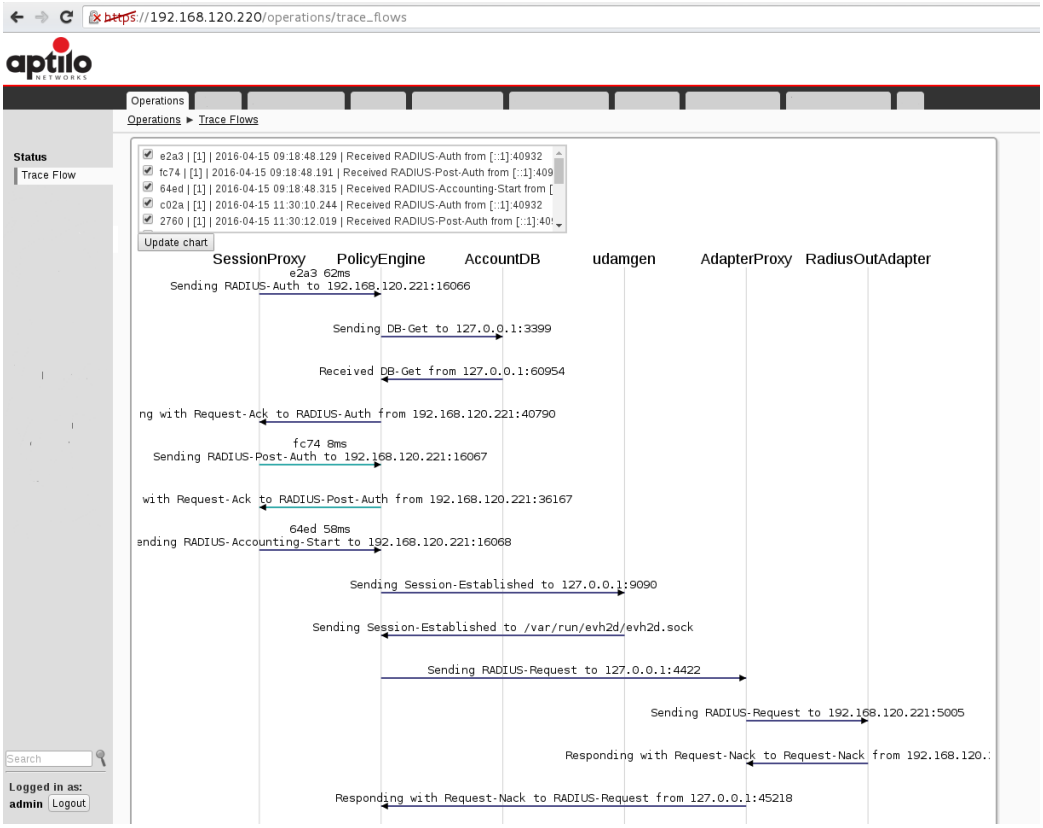


Figure 5.6: A view of the webpage when displaying a message or policy sequence diagram. The currently shown view shows a message sequence. The settings box above the diagram shows the last four characters of the UUID, trace number, timestamp, and the first message in the sequence for each sequence found. There are three different events visible in this figure. Events are shown further down the webpage if the user scrolls.

Figure 5.7 shows the flow inside the policy engine. This view is similar to the message sequence diagram, but some differences are noticeable. Since these messages are running on the same server the timestamps can be used to directly measure the time taken by each function. This means that above each function call the execution time taken for that specific function is shown. It is also possible to determine returning arrows because, unlike when dealing with the message sequence flow, each function returns. Similar to the message sequence diagram the arrows are clickable to access the log and each UUID to be displayed can be selected above the graph. If the list of UUIDs are very long the user might use the browsers built-in search function (Ctrl + F in most browsers) to simply search the list for the relevant UUID but since the list is order chronologically with timestamps it is also easy to simply scroll it to the relevant UUID. The length of the list depends on the amount of messages being traced in the system, and how long time it was since the log file was emptied, but may contain up to a hundred messages in a typical use case. The administrator can clear the list by moving the current log to the history folder since the list only displays the current log file.

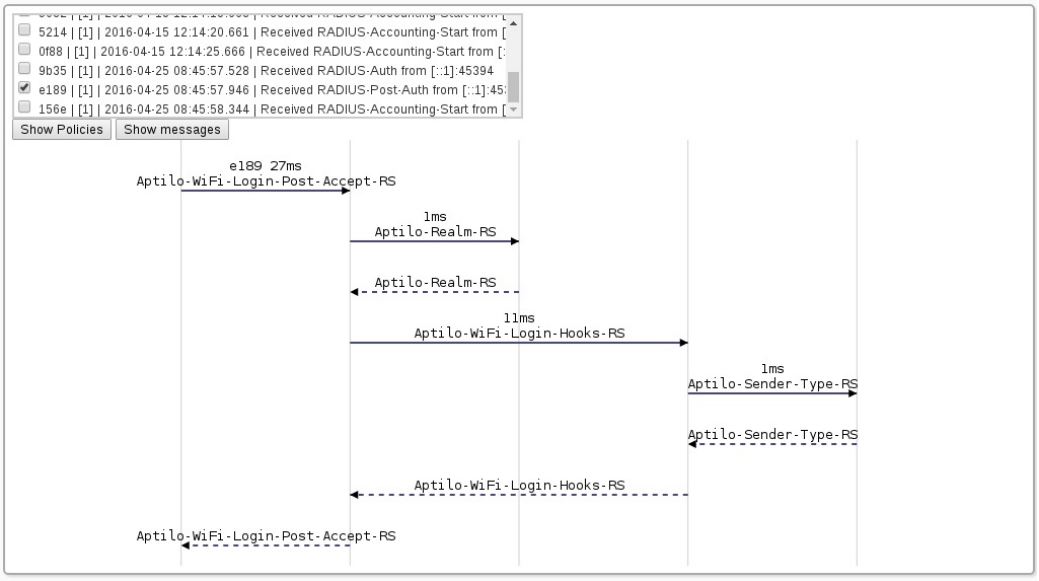


Figure 5.7: Example of a sequence of function calls inside the policy engine. The labels above are either of the function being called or the function it returns from depending on if the line is solid or not.

5.2 Results from the artificial neural network

The neural network created used the value 40 for both H and W resulting in a first layer with 1600 neurons. The value of 40 was chosen as the highest possible value that the host computer could handle, higher values crashed the Neuroph Studio application. The two hidden layers had 12 and 8 neurons respectively and the error tolerance during training was set to 1%. The setup of the neural network is further described and illustrated in Chapter 4 starting on page 47.

As seen in Figure 5.8 the neural network judged the graphs to be roughly 90% similar to good graphs. Unfortunately, since the network was only trained with 30 samples, these values alone cannot be used to determine the overall quality of the generated graphs and no statistical improvement can be measured. 30 samples can be compared to the values found in the UCI Machine Learning Repository[28] where 350 different data sets for machine learning can be found, and the median number of samples in the sets is 606. If we assume that the accuracy of the neural network follows a standard deviation, four times as many samples are needed to double the accuracy. This means that for any the top half of values in Figure 5.8 error bars to not overlap the lowest value's error bars, 74 more samples are needed.

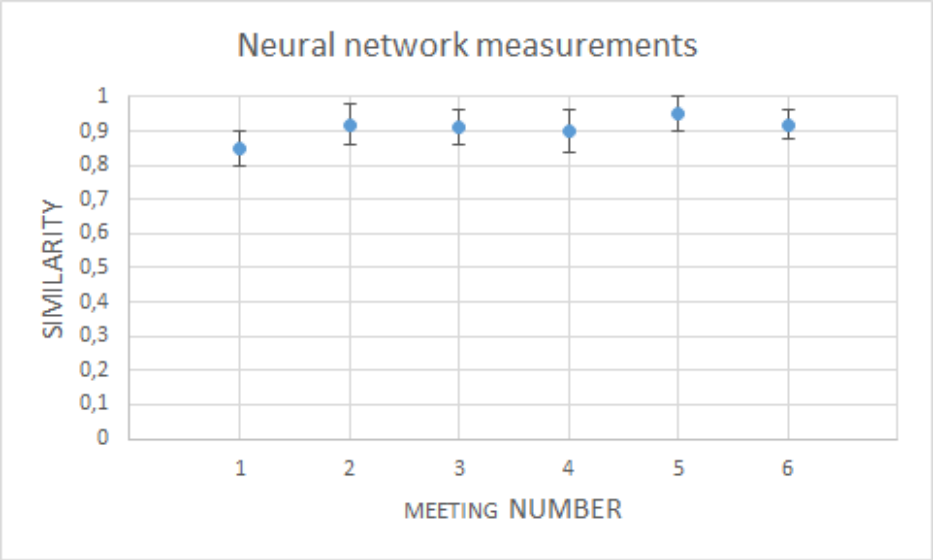


Figure 5.8: Graph showing the results from the neural network measurement done before each demo meeting. The value represent how similar the graphs generated were to the graphs the neural network had trained on. The error bars shows a 95% confidence interval.

Chapter 6

Analysis

This chapter will analyze the results presented in the previous Chapter. First the images presented will be analyzed and then the result from the artificial neural network will be analyzed.

6.1 Analysis of the graphs inside the web GUI

The layout presented gives the user two or three options of how they wish to represent properties. Properties that a graph only has one of, such as physical location, can be represented as both a complete graph, via the layout engine, or as an area behind the nodes with these same properties. The properties that a node may have multiple of, such as application group or subnet, can be represented as either a complete graph, or via the layout engine. The layout engine visualize the properties by grouping nodes sharing a property closely together. For instance, if the layout engine visualize application groups, the nodes will be attracted to other nodes within the same application groups as them, and nodes sharing multiple application groups between them will be even more attracted. It is also possible to completely turn of all information regarding any given property of a system.

This series of prototypes leads to a tool that makes it easy to only show the relevant information. This is easily seen in Figure 5.2 on page 51 where the user of the tool can easily identify application groups spanning more than one site. Previously the user would have had to manually open up an application group, get the list of nodes belonging to the application group, and manually go through the configuration of each of the nodes in the list to see if they are located at different sites. Now the user can get the same information by just clicking three buttons (”Group by: Application Group”, ”Link category - Application Group”, and ”Backgrounds - Site”) in the new graph tool within the web GUI.

The graphs presented follow the colors specified in ”Exploring Color in Interface Design”[25] and have a low number of edges crossing if the layout is set to group by the same property as the edges show. If they are set to present two different properties (such as layout done on application groups, while the edges show the subnet) the layout engine factors both in equal part. Since the user can customize the layout of the graph, as well as the areas and edges present in the graph, there may occur graphs which are suboptimal in their design, but this is hard to avoid due to the many different factors interacting.

6.2 Analysis of the sequence diagrams

The sequence diagrams follow the UML standard, but shows information relevant to the user, such as execution time. Each sequence diagram has an UUID to identify it so the user can look in the logs to get further details about what is happening in the system for a certain period of time. The user might also look at a series of events (i.e., a chain of events) by clicking on the checkbox for more than one UUID. Since the UUID is shared between the policy sequence diagram and the message sequence diagram, the user might look at both the policy execution and the message flow for the same event to see if all policies triggered upon the correct messages and in the correct order.

Realizing all the properties of the sequence diagram mentioned in the previous paragraph lead to an easy to use tool which translates complex logs, that previously were the best means to know what is happening in the system, into easy to read sequence diagrams. If the user wishes to examine the log for a specific event the user can click on the arrows in the sequence diagram and the log data will be presented on page within the browser. The new page contains the log with a customized search done. The search filters the log to only show the data for the clicked event.

6.3 Analysis of the artificial neural network’s result

The artificial neural network outputs a coarse result of how well the generated graphs match the graphs in the training set. Unfortunately, this result alone cannot determine whether the graph generated is good, but it ensures a more objective measurements than the meetings with the staff at Aptilo. Since the result of the artificial neural network were close to 90% for all graphs, we might consider all the generated graphs to be good graphs. Unfortunately, due to the low accuracy of the neural network, between the first and the last iteration of the prototype, any

progress in producing better graphs cannot objectively be measured by this neural network.

Chapter 7

Conclusion

This thesis shows a way to display a network with multiple properties displayed at the same time along with the ability to display UML sequence diagrams of both messages and policy sequences.

The thesis uses the Sigma.js Javascript library to provide an interactive webpage within the ALE framework's configuration utility. The system architecture view webpage provides a system administrator with an overview of the system's current status. The view can be customized to show a subset or combination of many of the properties of the system. Additionally, it is easy for the administrator to configure their own view of the system, so as to provide the information that they need to perform their own tasks. The user's interaction with the webpage consists of setting the layout to display a certain property, hiding, or showing different properties. It is also possible to click on specific elements to show their properties or 'Alt' click elements to access their configuration page.

The tool implemented in this thesis project also provides means to generate a sequence diagram of messages flowing between nodes in the system, as well as a sequence diagram displaying how the rulesets inside the system's policy engine were evaluated. This view also shows how long each function takes to execute. This makes it easier to identify those functions taking extraordinarily long to execute. Each arrow in the sequence diagram can be clicked to display the system log for that period of the specific function call's in the case of rulesets or the duration of a message flow in the case of message sequences.

All of the (sub-)goals in Aptilo's task description with the key words MUST and REQUIRED were completed and all but one of those with the keyword SHOULD were completed. The only (sub-)goal that is incomplete was the last one:

It SHOULD be possible to control the level of detail seen [in the UML sequence

diagram], for example, by limiting the nesting level.

Achieving this goal is left as part of future work.

Aptilo's impression of the work conducted in the thesis was that it was good and Aptilo's verdict on the work as presented is:

"The graphical overview will provide a simpler way for a administrators and support staff to view the current network configuration. The message sequence chart and policy sequence chart provide a clear view of system behaviour. Combined, these tools will reduce handling time for support cases and decrease the number of cases that require escalation. Additionaly they will be used during development and testing to automate the documentation of use cases."

The completion of the (sub-)goals combined with the response to the feedback given by Aptilo during the project means that the prototype of the tool is ready to be integrated into Aptilo's ALE framework. This integration is scheduled to be part of the ALE version 5.1 release in 2017.

7.1 Discussion

Evaluating the solution was harder than I had expected since I was unable to find a pre-existing tool that could evaluate the visual properties of a graph. Fortunately, the neural network and Aptilo's Friday meetings provided some feedback. Since the neural network's assessment is rather coarse, the statistical significance of the results are too low to determine anything except that the result presented is considered to be somewhat good. The meetings ensured that the work progressed and gave valuable input and output to the company, but the subjective assessment given in these meetings is hard to quantify in numbers and were therefore only used as guidelines during the work. This also means that the solution suggested in this thesis could use further evaluation.

Something that also took more time than expected during the work was adjusting to the coding practice at Aptilo and integrating with their existing code base. The use of "codeinspects" and strict git structure ensures a low technical debt, but was hard to adjust to as I had no experience with them from my previous studies.

7.2 Future work

Displaying more information in the different views could further enhance the product. Possible information to be added might include:

- Completing the last SHOULD requirement from Aptilo, as described earlier in this Chapter.
- Adding the message sequence as an overlay to the graph of the system's architecture,
- Error information and statistics as an overlay to the graph of the system's architecture or the sequence diagram,
- Logs in relation to the policies in the sequence diagram,
- Ruleset code in relation to the policies in the sequence diagram,
- When viewing properties as links, add a ”colorblind mode” to aid people with color blindness,
- Combining the graph of the system's architecture with geographical data to overlay the graph on a map, and
- Combining the graph of the system's architecture with the structure of the physical network to more easily identify single points of failure.

Another possible future effort is to analyze the graphs generated by tool presented with SocialAction described in section 2.6 on page 16. If SocialAction does not function properly, another tool is needed. Such a tool would have to analyze properties of the layout of a graph, such as calculating the number of edge crossings, colors, edge angles, and symmetry. This could be a possible project for a future master's thesis since evaluating the visual properties of a graph were something that was hard to do during this thesis project. As a result I only had a variety of guidelines to follow during it.

Bibliography

[1] "Aptilo Networks - carrier solutions for managing mobile data services." [Online]. Available: http://www.aptilo.com/

[2] T. Rybing, "Aptilo Networks." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Aptilo_Networks&oldid=717397730

[3] J. Wakefield, "One wi-fi hotspot for every 150 people, says study," *BBC News*, Nov. 3, 2014. [Online]. Available: http://www.bbc.com/news/technology-29726632

[4] "Aptilo Service Management Platform | Aptilo." [Online]. Available: http://www.aptilo.com/aptilo-service-management

[5] C. Steinbach, "Interview with Chris at Aptilo Networks," Stockholm, Sweden, Jan. 2016.

[6] "Java Neural Network Framework Neuroph." [Online]. Available: http://neuroph.sourceforge.net/

[7] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, USA: Oxford University Press, Inc., 1995. ISBN 978-0-19-853864-6

[8] C. Bishop, *Pattern Recognition and Machine Learning*. New York, USA: Springer, Oct. 2007. ISBN 978-0-387-31073-2

[9] J. Hua, M. L. Huang, W. Huang, J. Wang, and Q. V. Nguyen, "Force-directed Graph Visualization with Pre-positioning - Improving Convergence Time and Quality of Layout," in *2012 16th International Conference on Information Visualisation*, Jul. 2012. doi: 10.1109/IV.2012.31 pp. 124–129.

[10] W. Dong, F. Wang, Y. Huang, G. Xu, Z. Guo, X. Fu, and K. Fu, "An advanced pre-positioning method for the force-directed graph visualization based on pagerank algorithm," *Computers & Graphics*, vol. 47, pp. 24–33, Apr. 2015. doi: 10.1016/j.cag.2014.10.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0097849314001277

[11] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li, "Geometry-based edge clustering for graph visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 6, pp. 1277–1284, 2008. doi: 10.1109/TVCG.2008.135. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4658140

[12] C. Dunne and B. Shneiderman, "Motif simplification: improving network visualization readability with fan, connector, and clique glyphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 3247–3256. [Online]. Available: http://dl.acm.org/citation.cfm?id=2466444

[13] C. Dunne, "Measuring and Improving the Readability of Network Visualizations," *NIST ACMD Seminar, University of Maryland, College Park,*, Aug. 28, 2012. [Online]. Available: http://math.nist.gov/mcsd/Seminars/2012/2012-08-28-Dunne-presentation.pdf

[14] K. Ognyanova, "Static and dynamic network visualization with R," *Presentation from 8th Annual Political Networks Workshop & Conference (POLNET 2015)*, Jun. 18, 2015. [Online]. Available: http://kateto.net/network-visualization

[15] W. Dong, X. Fu, G. Xu, and Y. Huang, "An improved force-directed graph layout algorithm based on aesthetic criteria," *Computing and Visualization in Science*, vol. 16, no. 3, pp. 139–149, Nov. 2014. doi: 10.1007/s00791-014-0228-5. [Online]. Available: http://link.springer.com.focus.lib.kth.se/article/10.1007/s00791-014-0228-5

[16] C. Dunne, "Measuring and improving the readability of network visualizations," Ph.D. dissertation, University of Maryland, College Park, Maryland, USA, 2013, [Online] Available: "http://hcil2.cs.umd.edu/trs/2013-14/2013-14.pdf. [Online]. Available: http://gradworks.umi.com/35/99/3599522.html

[17] C. Dunne and B. Shneiderman, "Improving graph drawing readability by incorporating readability metrics: A software tool for network analysts," *University of Maryland, HCIL Tech Report HCIL-2009-13*, 2009. [Online]. Available: http://www-lb.cs.umd.edu/~cdunne/hcil/pubs/Dunne09Improvinggraphdrawing.pdf

[18] "SocialAction." [Online]. Available: https://sourceforge.net/projects/socialaction/

[19] "epistemex/cardinal-spline-js." [Online]. Available: https://github.com/epistemex/cardinal-spline-js

[20] M. Bostock, "D3.js - Data-Driven Documents." [Online]. Available: https://d3js.org/

[21] "Home :: Bugzilla :: bugzilla.org." [Online]. Available: https://www.bugzilla.org/

[22] "What is Kanban?" [Online]. Available: https://leankit.com/learn/kanban/what-is-kanban/

[23] "Extreme Programming: A Gentle Introduction." [Online]. Available: http://www.extremeprogramming.org/

[24] V. Visockas, "Comparing Expected and Real–Time Spotify Service Topology," Master's thesis, KTH, Royal Institute of Technology, Stockholm, Sweden, May 29, 2012, Trita-ICT-EX, 2012:63. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-96352

[25] H. Shubin, D. Falck, and A. G. Johansen, "Exploring Color in Interface Design," *interactions*, vol. 3, no. 4, pp. 36–48, Jul. 1996. doi: 10.1145/234813.234818. [Online]. Available: http://doi.acm.org/10.1145/234813.234818

[26] S. Åhman and M. Wallstersson, "EVH2 protocol : Performance analysis and Wireshark dissector development," Bachelor's thesis, KTH, Royal Institute of Technology, Stockholm, Sweden, June 30, 2012, Trita-ICT-EX, 2012:123. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-98689

[27] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978. [Online]. Available: http://dl.acm.org/citation.cfm?id=359563

[28] "UCI Machine Learning Repository." [Online]. Available: https://archive.ics.uci.edu/ml/index.html

Appendix A

Task description from Aptilo

This document outlines work required to help visualize the ALE system configuration and behavior.

A.1 Introduction

A typical ALE installation consists of multiple nodes; upstream and downstream nodes that are external to the ALE system, and nodes belonging to the ALE system. Each ALE node can be configured with multiple interworking components for external communication, policy evaluation, usage data and accounts.

Understanding the ALE system configuration, at a high level, is a prerequisite to working with an existing customer installation. Configuration information is made available via the ALE management interface, but is difficult to assimilate, and easy to misinterpret since it is spread out over many configuration pages.

The ALE system behavior is also highly configurable. In particular, the flow of control in the ALE policy engine is sensitive to changes in configuration and to differences in external stimuli. During trouble-shooting our understanding of the flow is informed both by familiarity with the policy rules, which may be incomplete, and reference to trace logs, which are dense and difficult to follow.

A.2 Requirements

Keywords ”MUST”, ”MUST NOT”, ”REQUIRED”, ”SHOULD”, ”SHOULD NOT” and ”MAY” that appear in this document are to be interpreted as described in RFC2119.

A.2.1 ALE System Visualization

The ALE management interface MUST provide a consolidated, schematic high-level overview system configuration including ALE nodes, and external nodes, node groups and networks. All nodes MUST be annotated with their configured names. The constituent external nodes of external node groups MUST be represented. For ALE nodes, membership in an Application Group MUST be represented. So too MUST allocation of adapters to and ALE node be shown.

Given that some ALE installations will be complex, involving many nodes, it MUST be possible to control the level of detail see in the system overview. This MAY involve, for example, hierarchical views. Part of the work will be to investigate the best way to manage this. The system visualization MUST be integrated into the existing ALE management interface and MUST therefore follow the guidelines described in the ”Aptilo Developer Handbook”.

A.2.2 Overlays

Once a schematic system overview is in place, we MAY supplement the diagram by overlaying statistical information retrieved from the ALE nodes. ALE already collects statistic relating to request rate and timeouts for policy and adapters. Part of the work would be to investigate what statistical information might be represented and how.

We MAY also correlate errors and warnings in the system log with individual nodes. We MAY, for example, overlay nodes in the diagram with colors representing the number of severity of the log messages. Part of the work would be to investigate the best way to display this information.

A.2.3 Flow Visualization

It SHOULD be possible to generate diagrams exposing system behavior from user trace (i.e. from a text log of the message and policy control flow).

Augmenting the user trace MAY be necessary in order to make log interpretation feasible. Any changes to the user trace MUST NOT reduce its readability or, alternatively, it SHOULD be possible to disable the augmentation.

So that we may version control and include these diagrams as documentation, any tools created to generate diagrams MUST output a text-based source (e.g. the dot language as used by graphviz).

A.2.4 Message Flow Diagram

A tool SHOULD be designed and implemented to generate a UML sequence, or comparable, diagram showing the sequence of message flows between ALE nodes and components. The input to the tool will be system user trace and system configuration. Part of the work will be to investigate how best, and in what detail to represent nodes, components, messages and flow sequence.

Used in combination with network capture files, the message flow diagram MAY include messages sent to external nodes.

A.2.5 Policy Flow Diagram

A tool SHOULD be designed and implemented to generate a UML collaboration, or comparable, diagram showing the flow of control between policy rulesets. The input to the tool will be system user trace and, if necessary, system configuration. It SHOULD be possible to control the level of detail seen, for example, by limiting the nesting level.

As a possible extension the policy flow MAY include Lua function invocations. The Message Flow Diagram and the Policy Flow Diagram MAY be combined if the resulting diagram is still comprehensible.

Appendix B

Learning sources for the artificial neural network

[1] C. Dunne, “Measuring and improving the readability of network visualizations,” Ph.D. dissertation, University of Maryland, College Park, Maryland, USA, 2013, [Online] Available: ”http://hcil2.cs.umd.edu/trs/2013-14/2013-14.pdf. [Online]. Available: http://gradworks.umi.com/35/99/3599522.html

[2] K. Ognyanova, “Static and dynamic network visualization with R,” *Presentation from 8th Annual Political Networks Workshop & Conference (POLNET 2015)*, Jun. 18, 2015. [Online]. Available: http://kateto.net/network-visualization

[3] T. Masui, “Evolutionary Learning of Graph Layout Constraints from Examples,” in *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’94. New York, NY, USA: ACM, 1994. doi: 10.1145/192426.192468. ISBN 978-0-89791-657-8 pp. 103–108. [Online]. Available: http://doi.acm.org/10.1145/192426.192468

[4] P. Healy and N. S. Nikolov, *Graph Drawing: 13 Th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005, Revised Papers*. Springer Science & Business Media, Jan. 2006. ISBN 978-3-540-31425-7

TRITA-ICT-EX-2016:111