



Improved Network Map Display

Joel Oughton

This report is in partial fulfilment of the requirements for the degree of Bachelor of Computing and Mathematical Sciences with Honours (BCMS(Hons)) at The University of Waikato.

Contents

1	Introduction	5
2	Related Work	8
2.1	Research	8
2.1.1	Computer Network Visualisation	8
2.1.2	Web Based Displays	9
2.1.3	Published Implementations	9
2.2	Current Implementations	10
3	Background	11
3.1	Datasets	11
3.2	Technologies	11
3.2.1	Graphics Technologies	12
3.3	Libraries	15
4	Visual Design	16
4.1	Nodes	16
4.2	Edges	16
4.3	Layouts	20
4.4	Navigation	21
4.5	Overviews	24
5	Implementation	26
5.1	Client / Server Model	26
5.2	Nodes and Edges	27
5.3	Groups	28
5.4	Layouts	29
5.5	Navigation	30
5.6	Overviews	31
5.7	Overlays	32
5.8	Editor	33
6	Evaluation	35
7	Conclusions and Future Work	37

A KAREN PHP Network Weathermap	42
B Rural Link Nagios Network Map	43
C Evaluation Question Set	44

Abstract

Advances in web browser technologies have allowed for highly interactive, cross platform applications that have not been fully utilised in the computer network industry. The aim of this project has been to use these technologies to produce an improved implementation of a network map that effectively communicates topology and performance data to its users. The development of a map has involved assessing HTML5 technologies, identifying important features of network maps, researching relevant information visualisation theory and developing working prototypes. Representing computer networks in particular can be difficult when the number of nodes and links is large. For this project, a network mapping tool called NetMapJs was implemented to address weaknesses in current tools. Visualisation techniques were identified and used that were effective in addressing the scalability of displays, such as semantic zooming and layout grouping. This, along with an expressive way of displaying network bandwidth, has resulted in an improved presentation of network map displays. To evaluate the implementation, network maps for two operational networks were developed which received good feedback from the engineers in those organisations.

1 Introduction

Computer network maps are an effective way of visualising complex network systems. Network maps consist of nodes and edges, where nodes represent devices and edges show the physical or logical relationship between them. The positioning of nodes throughout a map displays the topology of the network that is being visualised. It is important to position nodes correctly because the maps may be used to gain insight into the shape and structure of network devices. Automatic layout algorithms may be used to attempt a suitable layout or users may want to define static positions for nodes when there is already some idea about the underlying topology. Also, the various users of a network map may need to view it from different perspectives depending on their role. For example, a support desk worker may only need to see whether or not very general areas of the network are functional, whereas a network engineer may need to see a specific area in greater detail.

In network maps, the edges connecting nodes indicate a relationship between two devices. Networks often monitor and store performance metrics of a link that can be directly related to a single edge. It is therefore possible to visualise the metrics as part of the edge graphics or to include them through user interaction. This idea assists with two uses of network maps: monitoring and dimensioning. Network maps can be used to monitor the health of the devices and links, and visually indicate to the user when a problem is detected. With an effective layout in place, areas of the network that are affected by a problem can be easily identified and passed on to engineers to investigate. Dimensioning in networks is finding the minimum capacity of a link to service the peak capacity [10]. Network maps which show bandwidth usage on links can assist network engineers with determining the peak capacity of links and whether there is more bandwidth available that can be sold to customers.

In this project, techniques and technologies are explored with the goal of producing an improved network map display over existing implementations. NetMapJs, a web based network map visualisation application, was designed and developed as part of this project to present the ideas discussed in this report. There are some notable weaknesses of existing network maps that this project aims to address:

- Lack of interactivity
- Scalability issues with large networks
- Specialised software requirements
- Limited visual display of quantitative information

Current network maps often generate static images that provide little or no user interaction. This project looks into the use of zooming, panning and other interactive navigation functionality to give users more control over the information space. A user-friendly event system encourages developers to

make use of user input. Clicking or hovering over nodes and edges, for example, calls event functions that can be customised to add to the visual display without needing an understanding about how NetMapJs works internally. Existing tools also typically do not scale well for large networks, as views become overwhelming and overlapping edges obscure parts of the network. This project considers the use of hierarchy to reduce the overall complexity of maps by grouping sub-networks together. The idea of groups coupled with semantic zooming enables detail to be abstracted until the user indicates that they are interested in a more specific area of the network.

Current computer network visualisation tools in the network industry do not take full advantage of the latest web technologies. Visualisations built and presented using HTML5 technologies in general are becoming more popular because the information can be presented in a web browser. This avoids the need to develop, maintain and install specialised software for multiple systems. Web browsers are ubiquitous so views can be easily shared and discussed between users. Mobile devices also come with browsers which gives web applications instant portability to other systems. The latest HTML5 specification provides a large step towards a completed standard [13]. This allows advanced web applications to be produced without the need of special proprietary software or plugins and will lead to improved cross browser compatibility through the standardisation of the HTML5. This project focusses on taking advantage of these technologies in order to produce a highly interactive computer network map that is easily accessible.

The performance of links between devices in a network coupled with an idea of topology is useful for capacity management and monitoring. A link may not be using all of its available bandwidth and so there would be a possibility of adding additional clients. The opposite is also useful; an indication of high bandwidth utilisation lets administrators know that more links or load balancing may be beneficial to their clients. Existing network map implementations sometimes attempt to display this information in edges. This is most commonly done using either the width line or a set of colours. However, the use of colour is one of the least useful ways to display quantitative variables [28]. In this project, a new way of displaying quantitative data within edges that uses edge width, length, orientation and colouring is presented and implemented in NetMapJs with good results.

The datasets used in this project for testing NetMapJs were sourced from two real and currently operational networks, Rural Link and KAREN. Both networks have mapping tools currently in place. A brief comparison of these tools with NetMapJs is included in this report. In the evaluation section, network engineers from within Rural Link and KAREN were also asked to compare the tools and provide feedback.

This report is divided into the following sections. Section 2 gives the necessary background information used for the rest of the report. In Section 3, the datasets are described in detail and the

technologies and libraries used in this project are listed. The visual design considerations are described in Section 4. Each of the design aspects are motivated, justified and an example from NetMapJs is given. The technical details about how the visual design was implemented are found in Section 5. In Section 6, the results of evaluation by network engineers are discussed. Finally, conclusions are drawn and some possible directions of future work are mentioned in Section 7.

2 Related Work

In this section, work relevant to this project is discussed. The work is divided into two categories, research and current implementations. In Section 2.1, published literature focussing on web based visualisation, network graphing issues and published implementations is analysed. Section 2.2 looks at some of the popular network map implementations and how this project attempts to address the weaknesses of them.

2.1 Research

The research related work to this project can be divided into three areas: issues relating to computer network visualisation, web based displays and actual network map implementations.

2.1.1 Computer Network Visualisation

Computer network visualisation work includes the related key issues such as presentation ideas, techniques to address network scalability problems and interactivity considerations.

Eick describes aspects of network visualisation and identifies strengths and weaknesses of graph based displays [6]. Network maps are most commonly visualised using node and link graphs. It is noted that these maps are particularly effective for small and sparse networks. Problems arise with larger networks such as display clutter, node positioning difficulties and perceptual tension. Display clutter is the situation where displays become overwhelmed with too much information. This may cause the viewer to become confused with the display. Node positioning affects the way that viewers interpret the map. Different node positions may lead to different interpretations of the same underlying data. Finally, the distance between nodes cause viewers to perceive relationships between nodes differently. For example, nodes positioned closely will be joined by a short line and will appear to be related to each other. An effective network map layout takes advantage of the perceptual tension effect.

Eick presents three strategies to address these problems: dynamic parameter focussing, node positioning and the use of a 3D layout. This project considers the first two but only uses 2D layouts. While it is possible that the visualisation techniques that are presented could be applied to a 3D layout, it is considered outside the scope of this project and left to future work.

Becker, *et al.* examine the limitations of static network maps and explain the benefits of adding dynamic features [1]. They recognise the difficulty of deriving good heuristics for determining visual parameters such as line width and node sizing. Dynamic maps have the advantage of incorporating sliders or other controls that allow parameters to be adjusted and viewed in real time. They recognise that it is often useful to follow paths through a network. This becomes difficult for larger, more cluttered networks using a static map. The design of the network map for this project considers

these advantages as well as the benefits from being able to utilise the latest interactivity technologies available in the browser.

2.1.2 Web Based Displays

Rohrer and Swing saw the benefits of web based visualisations back in 1997 [27]. They state that the loose coupling between data, users and web applications is likely to provide a flexible medium for information visualisation applications. A side project mentioned in the paper is a web based network visualisation tool that uses translucency for detail abstraction and hyperlinks to explore nodes in more detail. This tool was developed over 10 years ago and does not include the advanced HTML5 features available today.

There are publications that consider ‘thin’ and ‘fat’ clients as the two extremes of client to server application interactions [7][15]. A thin client has most of the work done on the server side and a fat client is the opposite. Thin clients cause more load on the server and are limited in the amount of interactivity that is possible. A fat client, however, allows for sophisticated user interaction and imposes less load on the server. This project takes advantage of recent improvements to browser technologies by producing an application that primarily runs inside the browser. Only network data storage and processing is handled on the server side. The implementation of NetMapJs therefore falls between the thin and fat client types.

It is important to select a web graphics technology that is scalable as the number of nodes and edges increases in a network map so that larger networks can be visualised. Johnson and Kelly performed a scalability study on web-native information visualisation [16]. They test and analyse the scalability of the three most popular 2D web graphics technologies. SVG, Canvas and native HTML. The results show that Canvas performs the best out of the three technologies. The authors also note that neither SVG or Canvas perform well on large visualisation datasets. This paper was published in 2008 and browser support is anticipated to improve over time. Therefore as a part of this project it was deemed necessary to conduct a new set of tests, tailored to benchmark network visualisation in the browser.

2.1.3 Published Implementations

Paul, et al. produced a Java 3D implementation of a network weather map and a performance data poller [23]. The map incorporates an idea of subnetworks having specific layout types such as star, ring and sphere. More node detail is made available as the user moves through the 3D space. Complicated graphs displayed using three dimensions make depth perception difficult due to occlusion. Movement through the space is helpful for interpreting such displays. The Java 3D implementation suffered from low frame rates that reduced the effectiveness of the movement.

Becker, *et al.* present a geographical network mapping tool [2]. The visualisation is static with controls located around the map to look at different aspects of the data. They struggle with ideas to manage navigation when zoomed in. In this project, stacked overview maps are used to overcome this problem. They also look at ways of showing a large network in a compact display through aggregation of links and geographic omission. This is effective at reducing the display clutter resulting from too many overlapping edges. However, aggregation decreases the information about particular links. If the links are not disaggregated when zooming in, some of this information will be lost. This tool demonstrates useful concepts but is quite dated, so there are many improvements possible.

2.2 Current Implementations

There are already implementations of web based computer network maps. The PHP Network Weathermap reads in common network data files on the server side and generates an image representation of the network topology [17]. The image that is sent to the client side includes some interaction through the use of HTML image maps but requires a complete regeneration for a change of view. This limits interactivity and makes it difficult for a user to effectively navigate into deeper subnetworks. A user must click on a given node group and then wait for the page to reload completely with a new display of the internal nodes. This project attempts to address these shortcomings by using techniques described in Section 4.

Another web based implementation comes as part of the Nagios network management system (NMS) [21]. This network mapping tool only supports networks laid out in a tree structure, does not support interaction of any kind and does not visualise performance data. Networks that include redundant, non tree like links are visualised by multiple paths to compensate. The map does have a way of visualising whether or not a link between devices is up or down by highlighting areas of the tree green or red. An advantage of visualisations that are integrated into network management systems is that no additional installation is required because all of the networking data storage and processing is taken care of.

The Zenoss NMS also includes networking mapping functionality [32]. This tool takes a different approach to the zooming and panning method used in this project. Filters let the user immediately jump to a view of the network centered around a particular device. The amount of the network shown is adjusted using a slider that controls the number of hops from the central node. The automatic layout algorithm used is also adjusted by a slider that alters the level of node repulsion. This approach works well for smaller networks and in cases where a specific area of interest is known. In this project, zooming and panning is presented as a more effective way of navigating the information space of large computer networks.

3 Background

This section includes discussion about the datasets, technologies and libraries used in the development of NetMapJs. Section 3.1 describes the networks that data was sourced from and the current tools that operators use for producing network maps. Sections 3.2 and 3.3 note the technologies and libraries used and the reasons behind why they were chosen.

3.1 Datasets

Example datasets were required to evaluate network map designs and to support test driven development in the implementation stages. For this project, two main datasets were used for this purpose. Both of the datasets describe real networks currently in active service and the networks have an existing network map tool. This is particularly useful for this project because we can be sure that we are getting a good representation of an actual network and allows for an effective evaluation by receiving comparative feedback from the network engineers.

The first dataset used was from the KAREN network [18]. KAREN is a high capacity network that links together education and research institutions throughout New Zealand. The network consists of point-of-presence (PoP) nodes strategically placed in regions in the North and South Island and with connections to Sydney and Los Angeles. Each PoP can be treated as a subnetwork of KAREN that may contain distribution devices, and connections to institutions and other PoPs. Performance data such as bytes or packets per second for a given device port is split over a series of round-robin database (RRD) files. Device information such as name and location are stored in a database and are also available on their online weathermap. KAREN currently uses the network map generation tool called PHP Network Weathermap. A snapshot of the map can be viewed in Appendix A.

The second dataset was sourced from Rural Link's wireless network based in both rural and urban areas across Waikato. The network core is located in Hamilton city and branches out across wireless access points in a tree like structure with some redundant links. The dataset obtained includes device information and relationships without performance data. Rural Link uses a network management system called Nagios that also includes its own network map tool. The Nagios tool produces a tree map layout that handles redundant links by replicating paths. Appendix B contains an example of the Rural Link network map.

3.2 Technologies

In order to explain the choices of technologies, it is helpful to have a general overview of the implementation of the visualisation. The implementation of NetMapJs was built to run mostly on the client side with only the network data processing done on the server side. The visualisation code and network

datasets are loaded from server to client side where the web browser then loads in the application and displays the network map to the user. NetMapJs allows interaction with the user as well as additional requests to the server side for extra data such as the latest bandwidth measurements. As described in Section 2, this lies somewhere between a thin and thick client design.

It is useful to separate code dealing with the raw network configuration and performance data from the visualisation. This means that when network data collection practices or data structures change the visualisation can continue to function without alterations. This project uses a generic graph data structure on the visualisation side which will support any type of map. The data structure simply defines a set of nodes each with their own set of adjacencies. Nodes and adjacencies both may have data attached which allows for the addition of any number of parameters. Raw networking data, perhaps stored in RRDs or other databases, can be processed on the server side by an adapter and converted into the generic form that the visualisation understands. The generic visualisation structure uses JavaScript Object Notation (JSON) which makes it trivial to transmit between the server and client side [4]. For example, to convert the Rural Link configuration files to the generic visualisation form, a python script was written to act as an adapter between the two formats.

The NetMapJs application is implemented using JavaScript as the client side programming language. It was chosen because it is implemented in all modern web browsers, it does not require any third party plug-ins to run and its popularity has led to advances in its efficiency and performance. The JavaScript code handles client to server communication, user interaction and the actual visualisation generation. Topology and networking data is exchanged with the server using JSON. The visualisation is displayed to the user through the use of native HTML, CSS and Canvas. HTML and CSS were a natural choice for laying out elements on a web page as browser support is native and the standards are well defined. The choice of the web graphics technology however required some deliberation.

3.2.1 Graphics Technologies

The two most widely supported and popular browser native technologies are Scalable Vector Graphics (SVG) [8] and Canvas [12]. SVG uses vector based graphics whereas Canvas is a bitmap system. SVG allows vector objects to be created and added to the HTML Document Object Model (DOM). User interaction such as mouse clicks can be easily added to one of these objects and vector sets can be changed. Canvas on the other hand requires the programmer to develop their own user interaction system and the entire canvas must be redrawn when any change is required. It would therefore appear that SVG was the most natural choice for network maps that require user interaction for nodes and edges. However, previous tests have shown scalability performance problems with SVG [16]. These

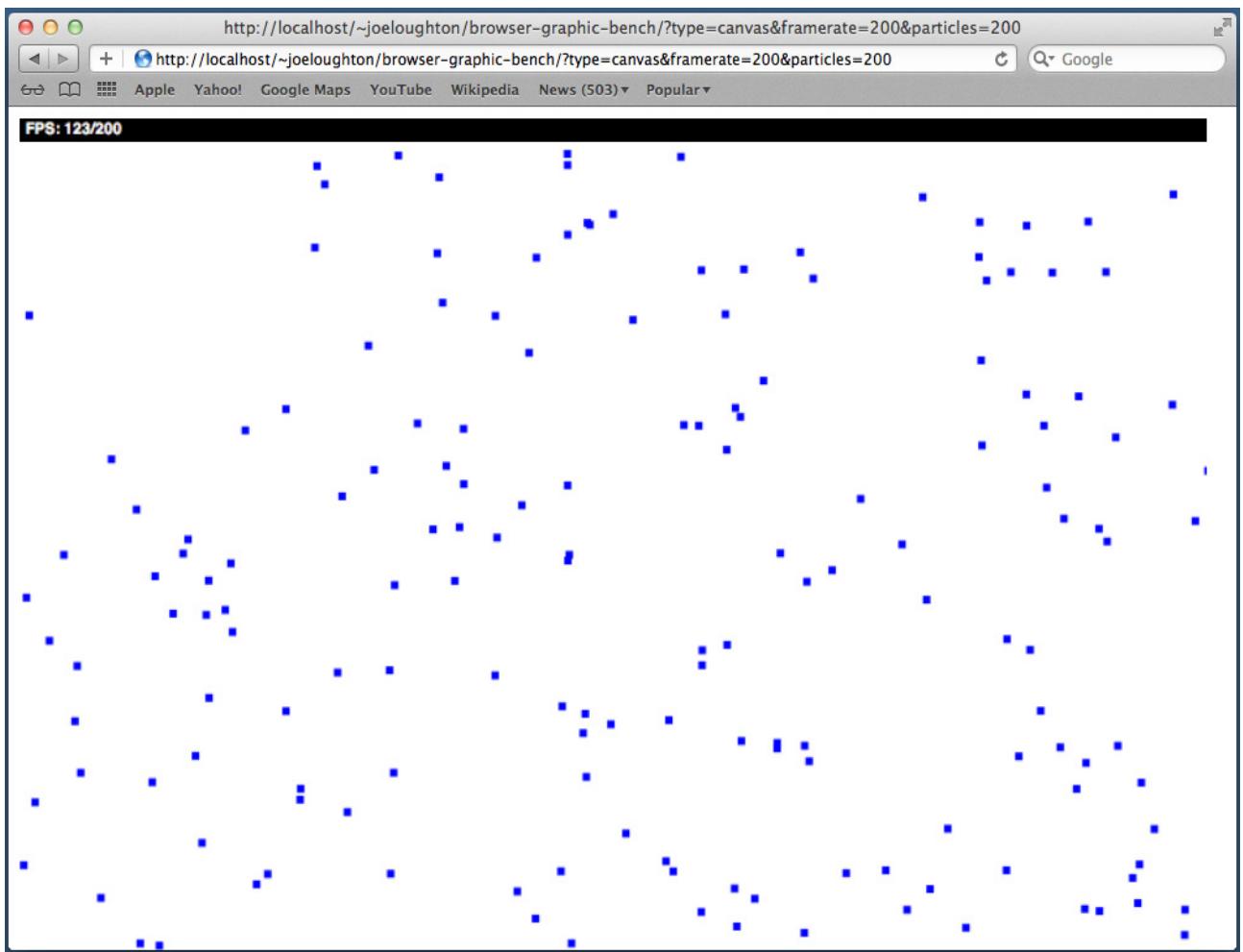


Figure 1: The browser graphics benchmarking tool currently rendering 200 particles using Canvas and running in Safari. The potential frame rate using this setup is shown to be 123.

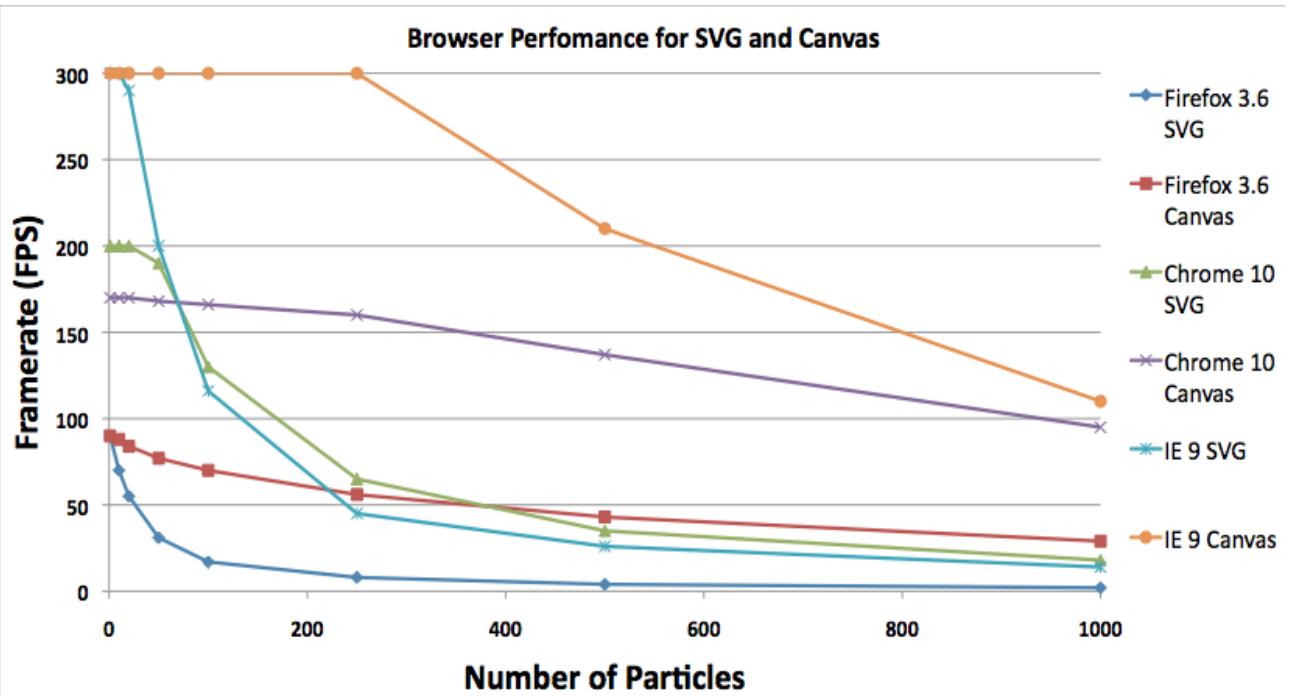


Figure 2: The results of the graphic benchmark run on a Windows 7 host. Three different browsers are tested using both SVG and Canvas.

tests are four years old relative to this project so it was deemed necessary to develop a benchmark tool to produce updated results.

The benchmark tool was developed using JavaScript to run across all modern browsers. It generates a given number of rectangles and begins moving them around the screen in random directions. The frame rate achieved for the current number of rectangles and the given technology is displayed at the top of the screen. See Figure 1 for an example of a running benchmark. This tool was run on Windows and Mac computers across the latest versions of the Firefox, Safari, Internet Explorer and Chrome browsers.

The results of the tests are shown in Figure 2. At the 500 particle mark the top three lines are all Canvas based frame rates. SVG performance is shown to drop significantly in the first 50 to 100 particles where as Canvas achieves a much more level gradient. As NetMapJs needs to be able to visualise networks that will potentially have many more nodes than the performance limits of SVG allow for, Canvas was chosen as the web graphics technology for this visualisation. However, the graphics generation system was kept separate from the rest of the code. This means that if the performance of these technologies shift in the future, a new graphic system could be implemented and slotted in to reflect the change.

3.3 Libraries

NetMapJs makes use of the jQuery, JavaScript InfoVis Toolkit and Arbor libraries in its design. JQuery was used for several reasons [26]. JQuery functions are portable and are especially useful for eliminating browser specific code. Cross browser support is important to ensure that each user gets the same experience across different systems. JQuery also simplifies HTML document manipulation, event handling and AJAX interactions. The JavaScript InfoVis Toolkit provides a framework for producing interactive, web based visualisations [3]. The toolkit includes many useful features which are needed for this project such as node-edge graph manipulation functions, complex number arithmetic and animation transitions.

NetMapJs includes some automatic layout algorithms for initially deciding where to position nodes in the network map. One of these layouts is a force directed algorithm. Force directed algorithms are a class of algorithms that lay out the graph in an aesthetically pleasing way by applying forces to nodes and edges. The Arbor JavaScript library was used for this as it performs well and provides mechanisms for changing physics parameters such as gravity and friction [29]. The QUnit library is a unit testing library plug-in for jQuery. This assisted with test driven development during the implementation stage of the project.

4 Visual Design

In this section, the most significant design aspects of NetMapJs are explained. Justification for each of the design decisions is given along with example screen shots taken from NetMapJs views.

4.1 Nodes

Nodes in network maps are traditionally used to express a particular physical or virtual device. Nodes are joined together by edges which represent the relationship or connection between them. Subnetworks are a specific group of devices that relate to each other by a common subnet. Subnetworks may consist of other subnetworks of their own and altogether they can be seen as a single network. It is possible to apply this concept to nodes when designing a visualisation. Nodes can visually contain sub-nodes for a given number of depth levels. Nodes that fall into this category will be referred to as group nodes.

Figure 3 shows the Hamilton (HLZ) PoP for the KAREN network. The outer circle marks the HLZ node and the smaller, solid circles inside illustrate the sub-nodes of that PoP. This grouped effect causes the viewer to assume that the sub-nodes are related to each other through a common parent node. Figure 4 shows a zoomed in view of the same HLZ group node. The parent node and edges are now faded out so that the focus is now on the sub-nodes and edges. This idea of zooming and adding detail is explained in Section 4.4. The layout of the sub-nodes within a group is also an important design decision as different group nodes may make sense to be laid out differently. The HLZ group node is using a star layout. Layout considerations are discussed in Section 4.3.

4.2 Edges

Edges between nodes in a network map indicate that two or more devices are connected to each other by some medium. A single line between two nodes is enough to show this relationship. If this line is thin then it reduces the amount of clutter for the same number of edges [30]. Figure 5 shows the display of a network with many nodes using thin lines.

In the case of network maps, there is often monitoring data available that directly relates to the performance of a given edge medium. This data could be accessed through a separate view in the visualisation but there are advantages to showing this directly on map. For example, if something goes wrong with a link then an indication on the main view on the visualisation will attract the user's attention immediately as opposed to the case where the user had to follow some interaction sequence to access the data. For this project, a new way of visualising quantitative variables describing a bidirectional network connection was developed. In particular, this method suits variables which have an upper limit such as the bandwidth or packet loss of a link.

Figure 6 gives an overview of how the connection visualisation works by using bandwidth as an

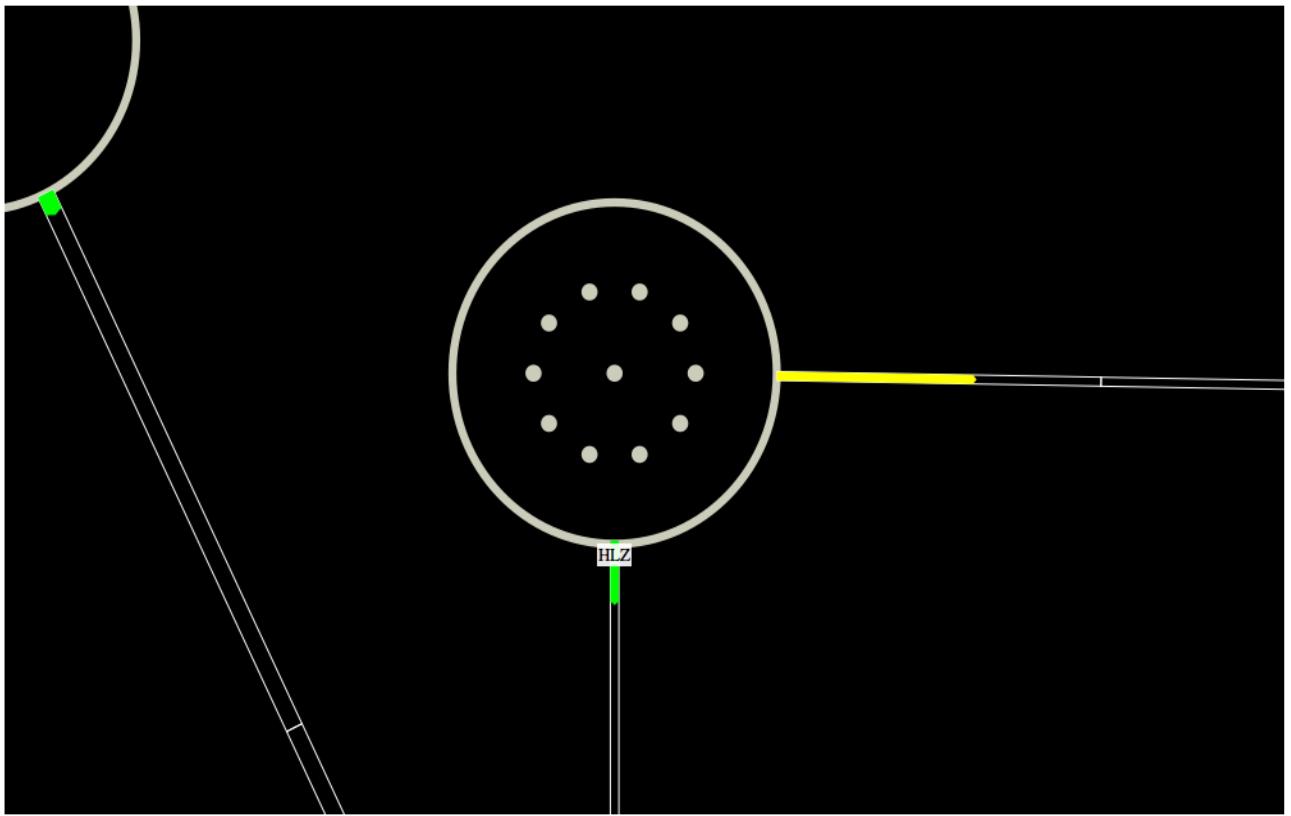


Figure 3: A section of the KAREN network map that shows the Hamilton (HLZ) PoP node.

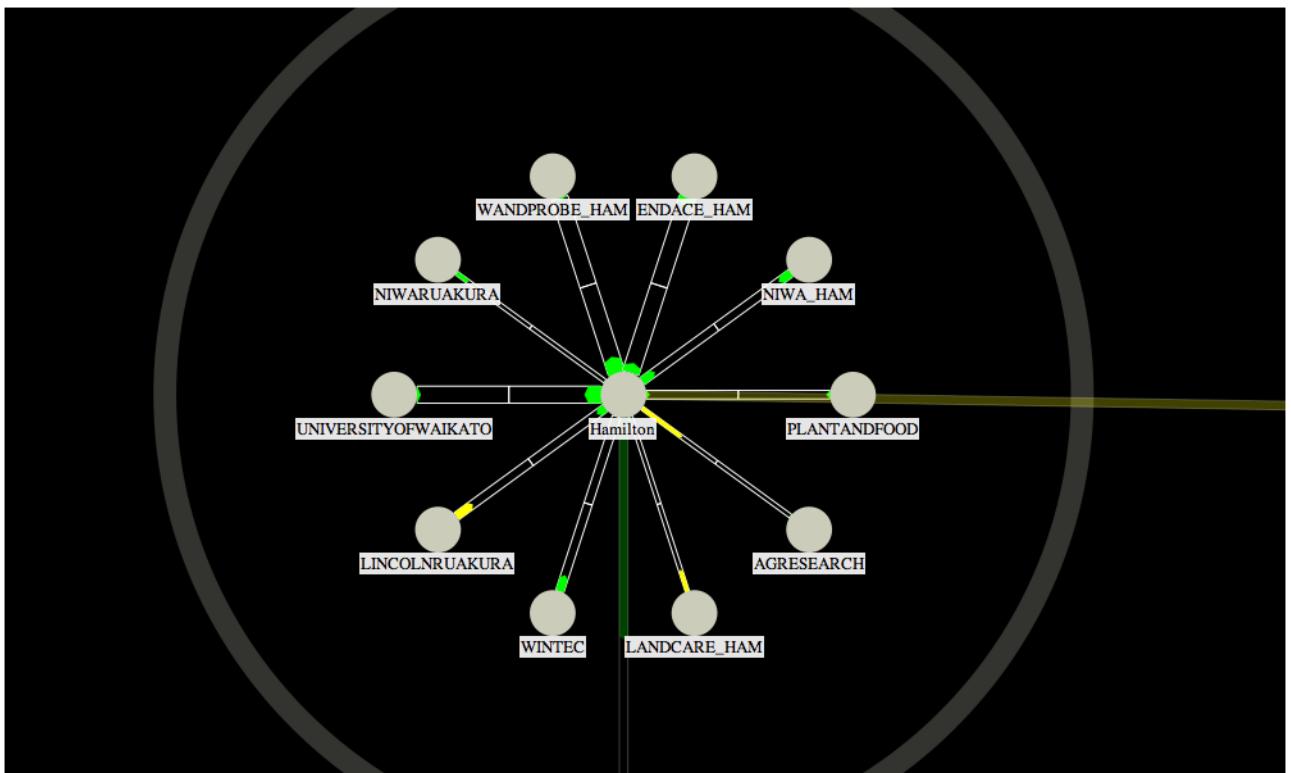


Figure 4: A zoomed in view of the HLZ PoP that shows its children nodes.

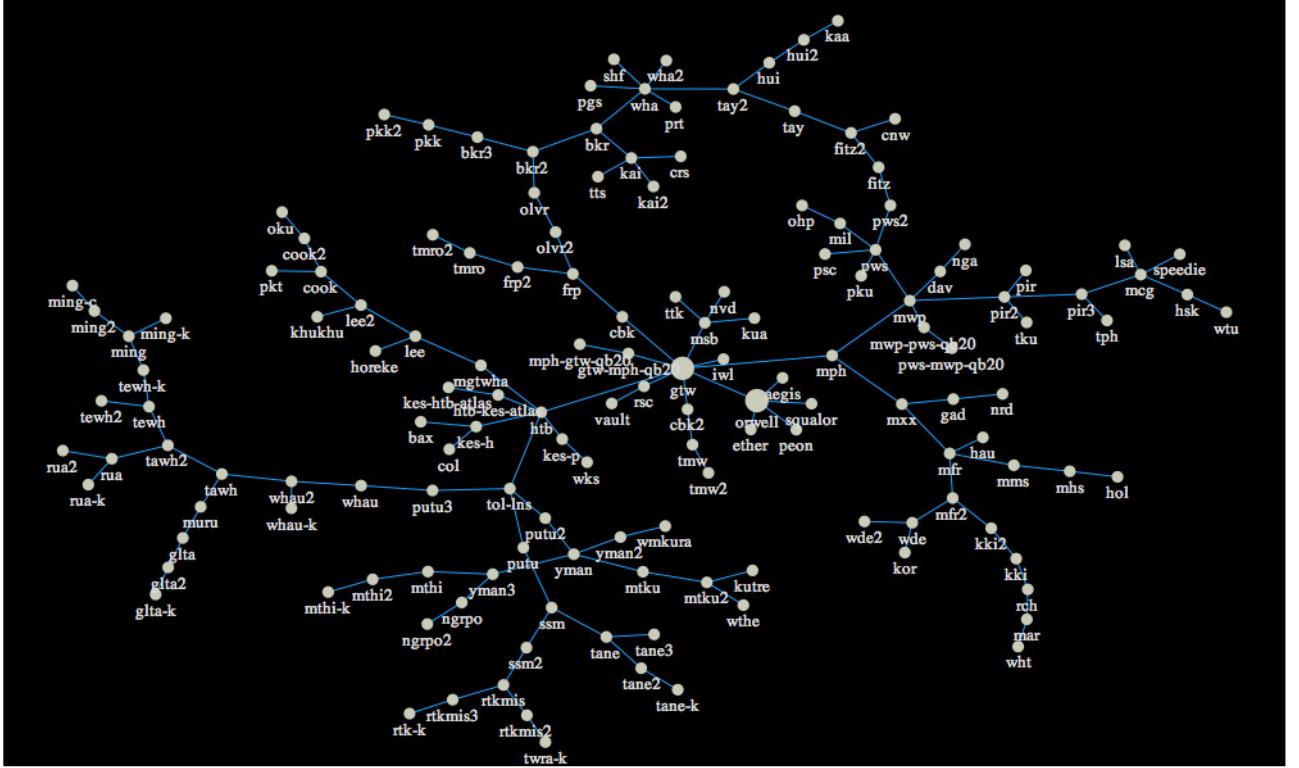


Figure 5: An overall view of the Rural Link network. Nodes are positioned to reduce edge crossings using a force directed layout.

example. A rectangle is drawn between two nodes and split at the mid point. The side of the split rectangle closest to a given node represents the direction from that node to the destination. The height of each of the rectangular sections indicates the total quantity of the variable. For bandwidth, this would be the total for that direction of the link and may be different to the other direction in the case of asymmetric connections. Arrows extend from a given node towards another with the length being relative to the percentage of the variable's proximity to its upper limit. The colour of the arrow is based on some user defined categorical variable such as the health of the network link.

The design for this graphic took careful consideration of information visualisation principles. Position and length are the most accurate methods for encoding quantitative variables [28]. The design uses length to show total and percentage utilisation. The use of position is also shown in the arrow as a proportion of the total rectangular section. Colour hue is effective at displaying categorical variables which is how it is being used in the visualisation. Figure 7 shows this design implemented in NetMapJs and is being used to show the bandwidth between the Palmerston North and Wellington PoPs in the KAREN map.

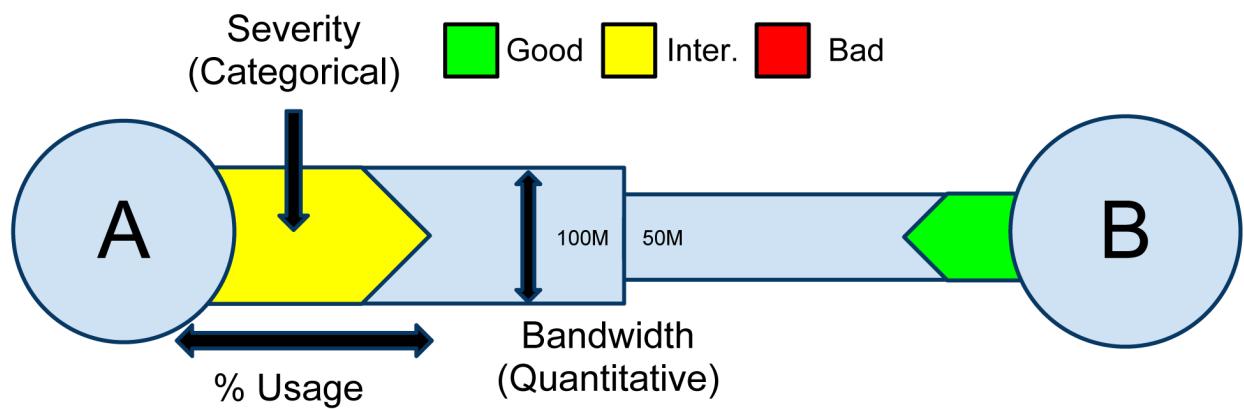


Figure 6: A diagram showing how edge data can be visualised effectively.

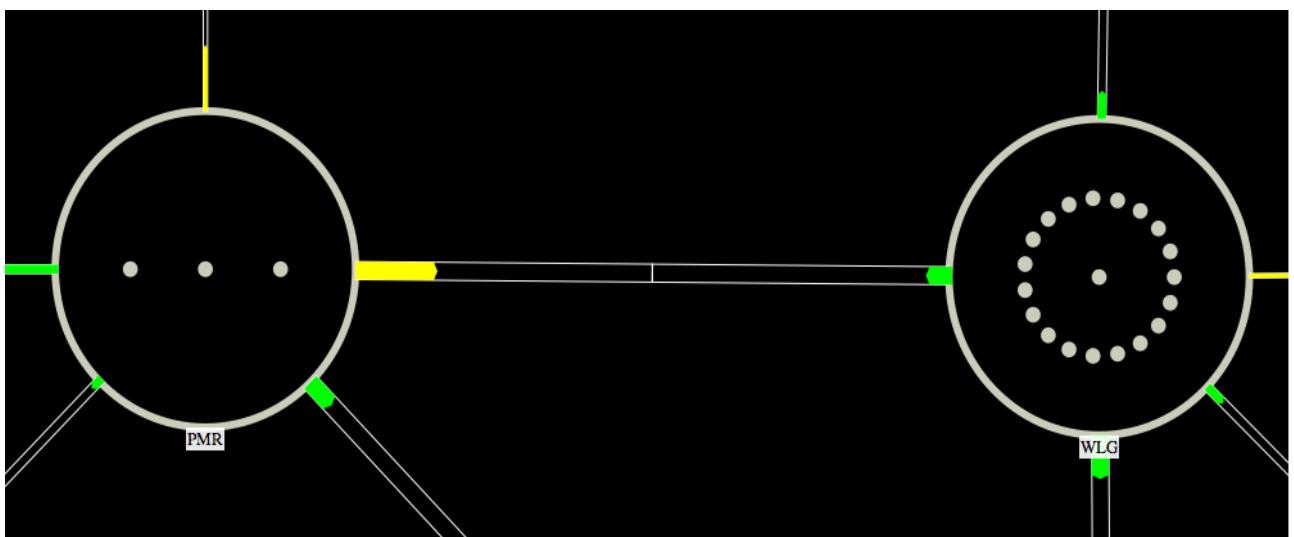


Figure 7: The new edge design shown within NetMapJs. The edge is connecting the Wellington and Palmerston North PoPs in the KAREN network.

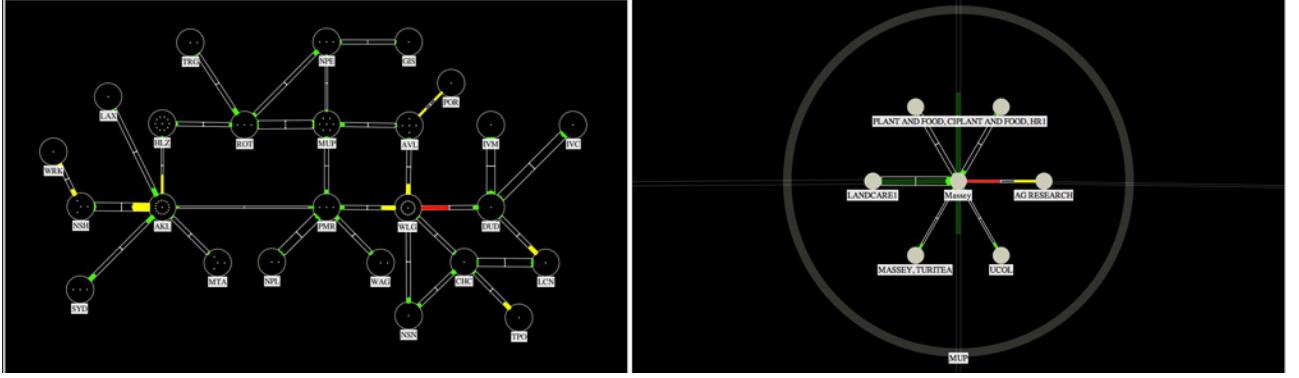


Figure 8: An overall view of the KAREN network is on the left that uses a static layout. On the right is a zoomed in view that shows the Massey sub group having a star layout applied.

4.3 Layouts

A good layout of nodes in a network map is important to maximise the insight and understanding possible to the user. A bad layout can lead to the cluttering of nodes and edges or, worse, a misunderstanding of the underlying data. The use of 3D layouts provide more available space for positioning nodes but can introduce new problems. Nodes and edges in a 3D layout may occlude other objects and it is hard to choose an ideal perspective within the 3D space [19]. 2D layouts are used in this project which simplifies the use of semantic zooming as described in Section 4.4.

Two basic 2D layout algorithms were used in this project in addition to statically defining node positions: force directed and star algorithm. These two were chosen because they were known to work well with the example datasets and provide a good starting point over purely static layouts. A complete network map tool would include many more layout algorithms to cope with a wider range of network layout structures [23]. Figure 5 shows Rural Link’s network laid out using the force directed algorithm. The algorithm lays out the network particularly well due to its tree like structure. In particular, there are no overlapping edges or nodes in the map. Redundant connections are shown clearly by branches that join up to form a loop.

Sub groups may make sense to have separate layout algorithm applied to them when the underlying network structures are different. An example of this is demonstrated in Figure 8. The overall view of the KAREN map contains statically defined node positions that are based on layout designs by a graphic designer. When the user zooms in to a particular PoP, it can be seen that the layout of that group is a star layout. One central node, the distribution router, connects to all the other devices in that PoP, making the star layout particularly appropriate for displaying subnetworks at the PoP level.

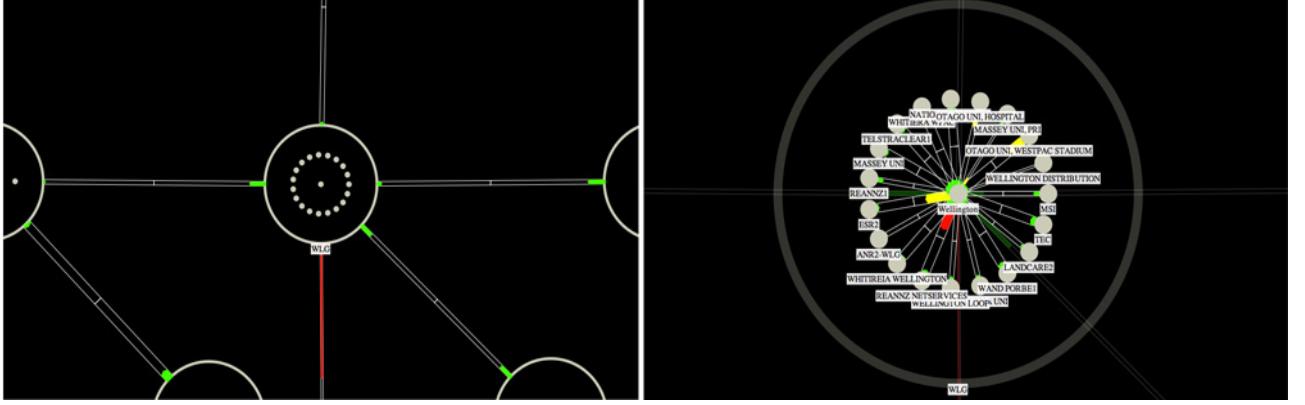


Figure 9: The view on the left shows an indication of more detail within the Wellington PoP. The zoomed in view shows the addition of more details such as edges and labels.

4.4 Navigation

Layout algorithms can be good for displaying large graphs clearly but there is a limit to their effectiveness. At some level of network size or density the limits of these algorithms will be reached and ideal layouts will no longer be guaranteed. It is also likely that a user can not easily decipher information and patterns from such large networks. It is therefore useful to combine layout algorithms with other techniques in order to remain scalable [11]. One way this can be done is through reduction techniques which decrease the amount of information being presented in the view. Grouping devices into subnetworks is one reduction technique that was previously described in Section 4.1.

Another technique for reducing networks is semantic zooming [25]. This is different from geometric zooming in that changes are made to the level of information displayed while moving towards a specific area of a graph. What would perhaps be visually overwhelming, if presented all at once, can instead be drip fed to the user as they indicate that they are more interested in a particular subsection of a network. NetMapJs uses semantic zooming along with panning to effectively navigate throughout large networks.

Figure 9 shows more detail being revealed to the user as they zoom in to a sub group of the KAREN network. In the fully zoomed out view it is important to indicate that there is distant content so that the user is aware that they can zoom in for further detail [28]. Sub groups indicate that they contain more detail by using small dots and, as the user zooms in to that group, more detail such as labels and edges between nodes appear. At this point, the group node graphic and its external edges become transparent which allows the user to focus more on the current group's detail while still being aware of the group that the nodes belong to.

While implementing the design of NetMapJs, it became apparent that it was increasingly difficult to navigate between nodes in the map as you zoomed deeper into sub groups. For example, when

zoomed into the Hamilton PoP in the KAREN map, it would require a lot of user interaction to view the directly connected Auckland PoP. Several drags of the mouse would be required to pan towards Auckland. To eliminate this problem, two navigation features were added.

The first feature enables the user to follow an edge to the node on the other end in one click. For example, if the user is currently zoomed in to the Hamilton PoP and wants to go to the Auckland PoP, they can simply click the edge that connects the two. An event triggered tooltip is displayed over an edge to notify the user of the node that lies on the other end of it. This is shown in Figure 10. The addition of animated transitions between view changes allow users to keep context and to absorb the change [20]. NetMapJs uses animation this way to transport the user to the new view such that they get a sense of where in the network they are moving to. A direct jump from the source to destination node risks confusing their perceived location in the overall network.

The other feature for improving navigation is zooming and centering nodes. Originally, when a user knows that they want to view a particular group node normally, they would perform a sequence of zooming and panning actions to bring it into focus. This feature allows the user to simply right click on a group node to automatically bring it into focus again using animation to change the views. Figure 11 demonstrates this effect. Combining the two features means that zooming and panning are not strictly necessary and allow the user to freely move around the network map quickly and effectively. The manual controls are still useful to make minor adjustments to the view and to zoom back out.

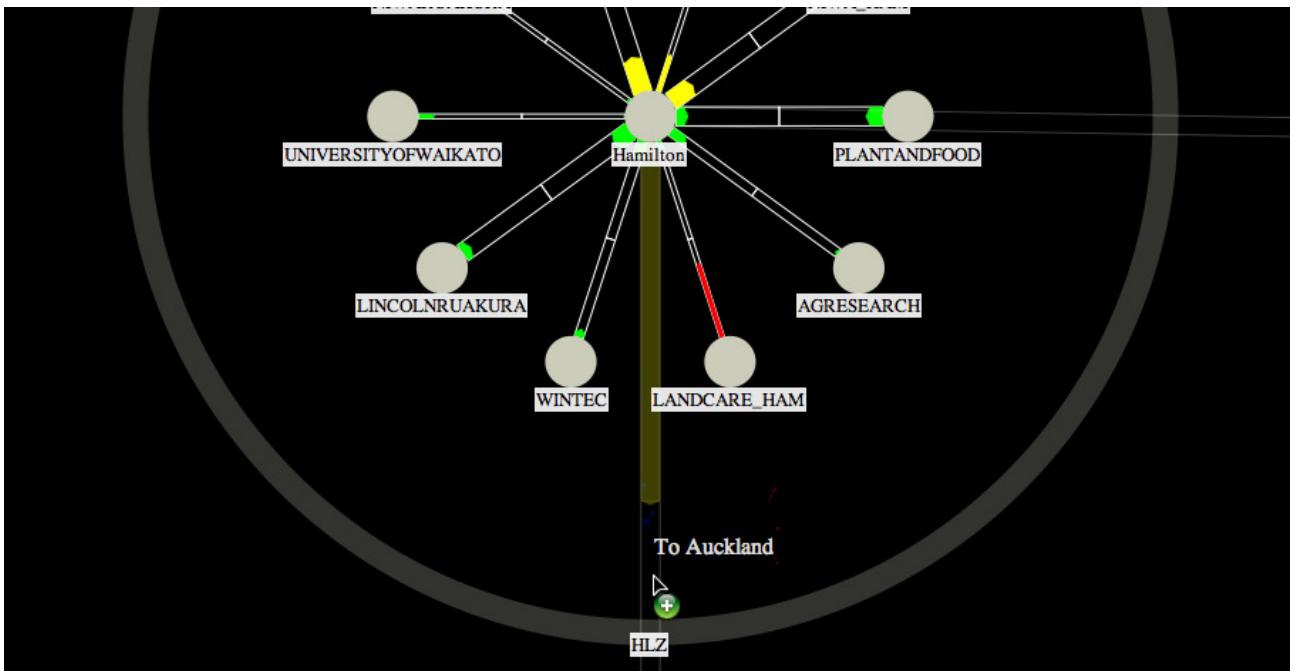


Figure 10: A tooltip is shown by the mouse pointer that indicated where the user will end up if they click on the edge.

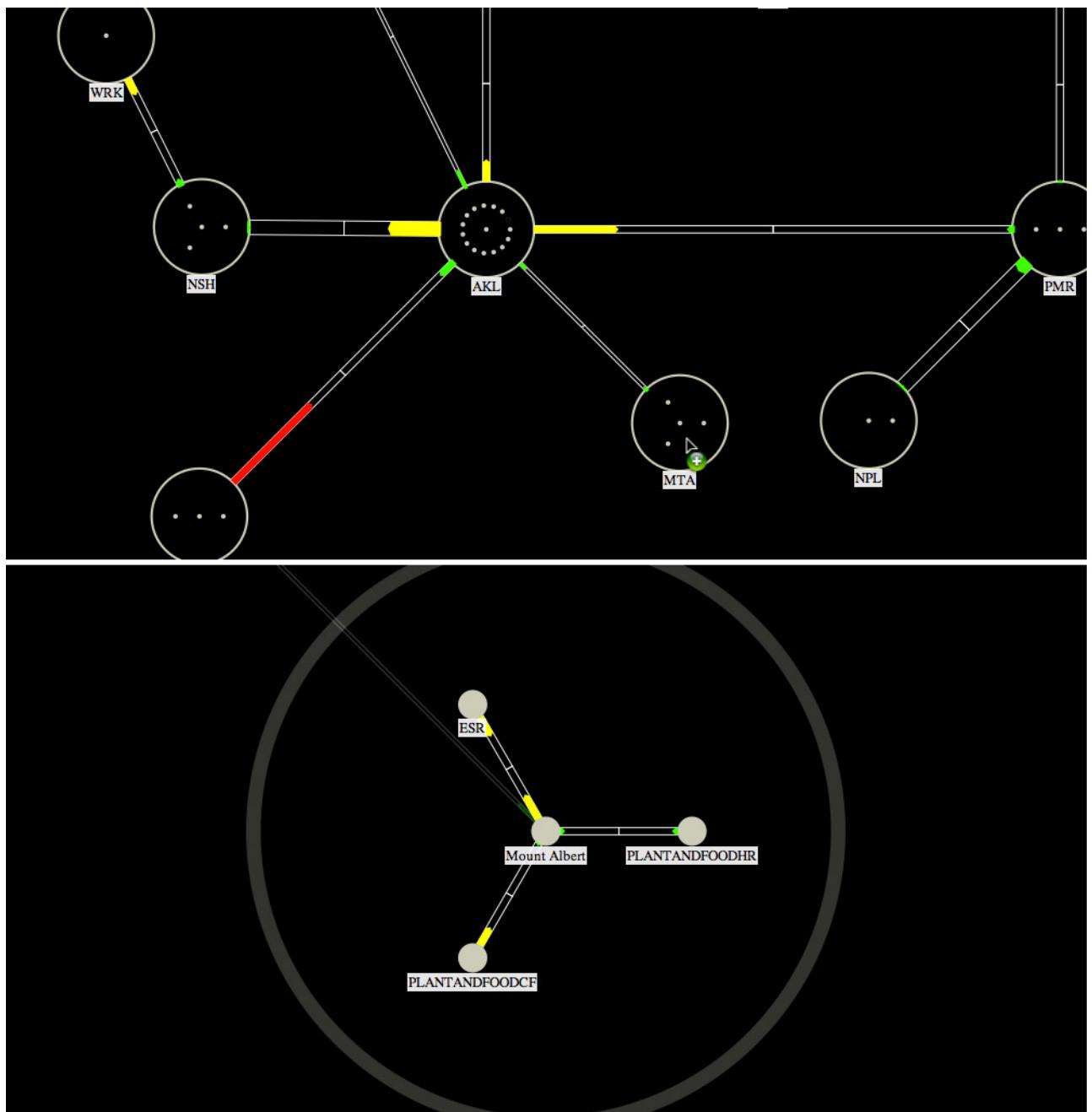


Figure 11: The top view of the KAREN map shows where a user is clicking. The bottom view shows where they would end up after the animated movement.

4.5 Overviews

As a user zooms in to deeper levels of the network map, it becomes easy for them lose context about where in the overall network they are. It is not ideal for the user to have to zoom out just to regain this context. To address this, NetMapJs makes use of multiple overview maps that each display a particular level of the network. A level is defined as a zoom range in which no detail is added or removed. Starting from an initial zoomed out view, as the user zooms into a sub group of the network, a separate overview map for that level is added to a stack of overviews. As the view port gets deeper into the map, more and more overviews are added into this stack. When the user zooms back out, the reverse happens and overviews are removed from the stack.

Figure 12 demonstrates this stacking effect. Also visible in the screen shot are yellow rectangles that have cross-hairs intersecting the middle. The outline of the rectangles show where exactly the user's view port is situated in the map as a whole as well as in each individual level. A cross-hair was used to make sure the view port location can still be identified at higher levels where the rectangle becomes too small to see. The overviews are also linked to the main visualisation such that the user can click and drag the rectangles to move the main view port accordingly. This allows for large movements across the map while in a zoomed in view.

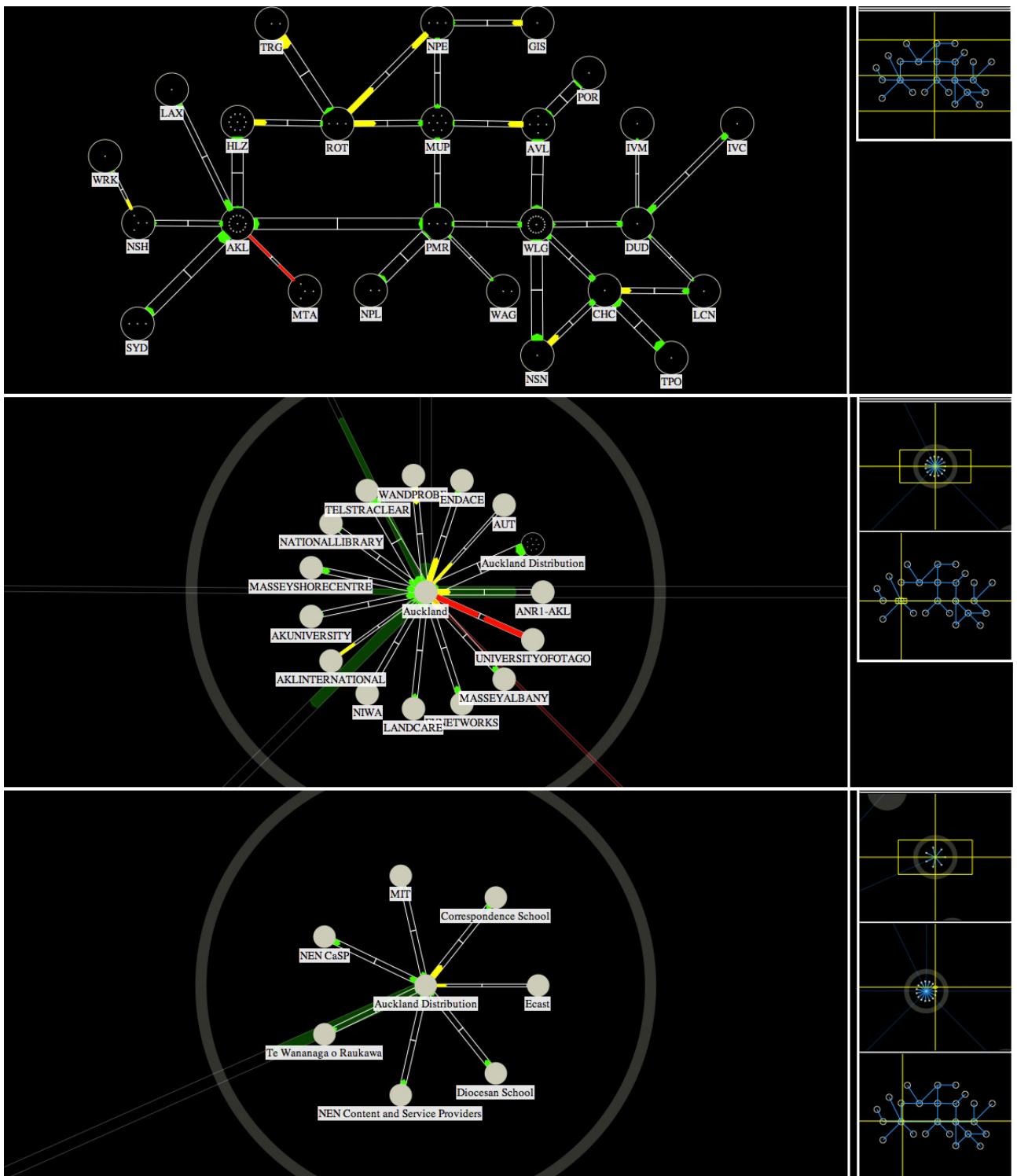


Figure 12: Three stacked screen shots of the KAREN map, each at different levels of zoom. At deeper levels, more overviews are shown to be added to the stack and the view port boxes change size and position accordingly.

5 Implementation

Details of the implementation of NetMapJs are described in this section. An overview of the tool is given in Section 5.1. The rest of the sections describe how aspects of the visual design were implemented. Supporting functions such as overlays and an editor are discussed in sections 5.7 and 5.8 respectively.

5.1 Client / Server Model

NetMapJs was implemented to run mostly on the client side with only some of the work done on the server side. Figure 13 shows the main components of both the client and server side, and how they interact to produce the final application. The NetMapJs application, shown in the client side, is loaded from the server and then run in the client web browser. Initial map layout data is then requested, visualisation data structures and events are initialised, and a display is generated for the user. At this point the user has full control to explore the visualisation.

The underlying network topology, performance values or other data may change over time. For example, bandwidth data for links is likely to change frequently throughout the course of a day. NetMapJs uses a data poller which is built using JavaScript intervals that periodically and asynchronously polls for updated data from the server side. This is done using AJAX methods to ensure that the visualisation remains responsive and can be updated without refreshing the browser page [24].

NetMapJs allows features of the maps to be manually adjusted, such as the node positions. In a purely client side JavaScript application, user state is lost when the web page is refreshed. To be able to make manual adjustments that can be saved and reloaded, there must be a server side function with privileges to read and write files to disk. When map configuration is saved, the NetMapJs JSON data structures are passed to a state storage utility. This utility handles the asynchronous transmission of the JSON data to the server. On the other hand, when configuration needs to be retrieved, the storage utility requests the JSON file from the server and proceeds to pass it back to NetMapJs where it will used to rebuild the network map.

In this project, the main focus was on designing and producing the NetMapJs client application. Only parts of the server side were implemented in order to create a working application. The server side in Figure 13 shows how the components would fit together if implemented completely. A web abstract programming interface (API) separates the client and server functions to allow for visualisation design freedom that is independent of web service constraints [31]. It also means that the client is dependent on the API calls rather than any given server implementation.

The server side application handles saving and loading stored network map states as well as

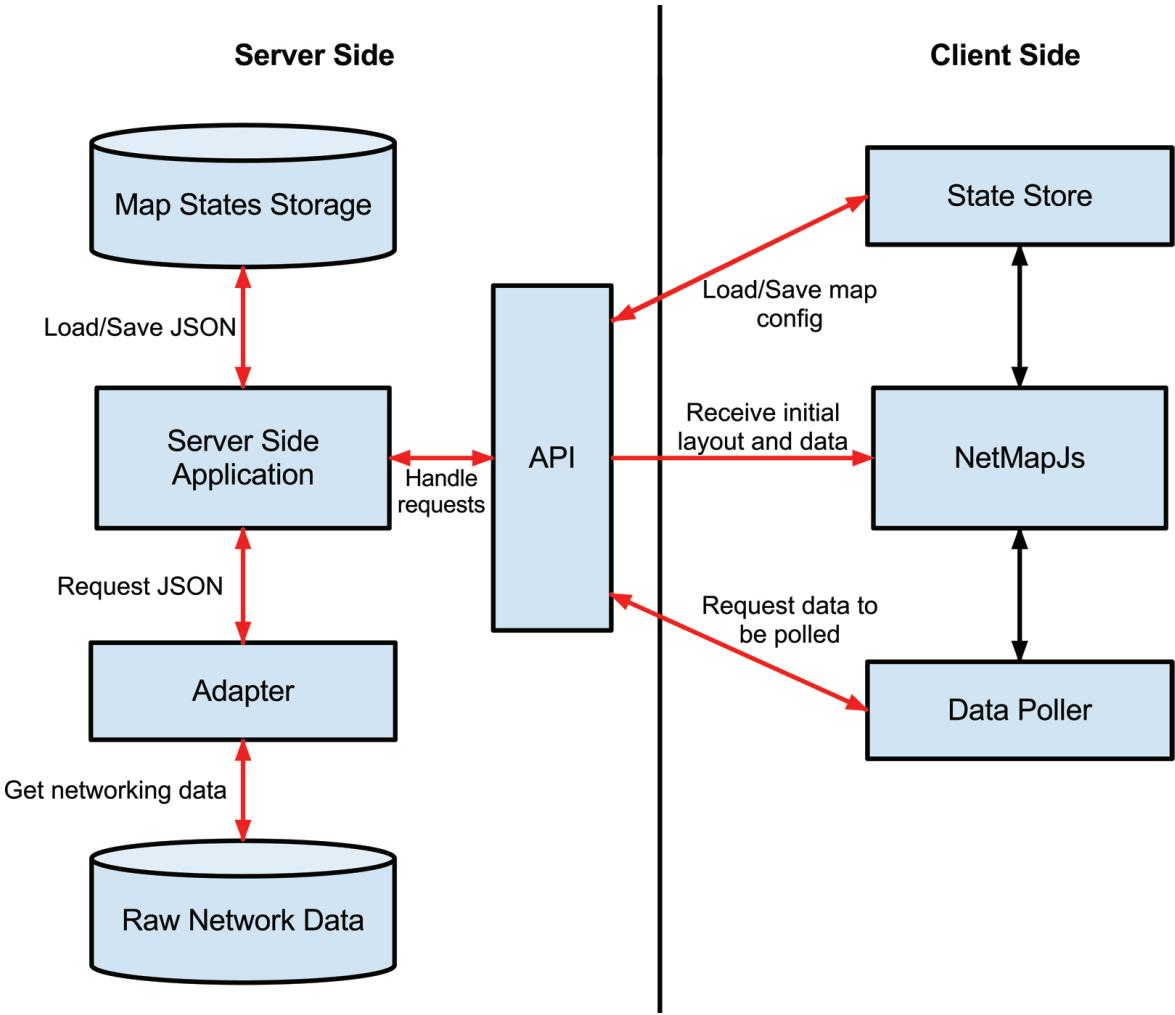


Figure 13: Overview of the client to server side architecture.

gathering raw networking data through interactions with an adapter. Adapters know about a specific networking data format such as RRD and how to transform this into the generic JSON format that NetMapJs understands. As an example, an adapter was developed for Rural Link that took their device configuration file and produced a valid JSON structure that could be read and visualised on the client side. The state and networking data is passed to the client through the use of the web API.

5.2 Nodes and Edges

The JavaScript InfoVis Toolkit has a built in system for handling the drawing of nodes and edges. They each have a set of helpers which are objects that include a display and contains method. The display method describes how a node or edge should be drawn to the canvas and is called each time the map needs to be redrawn. The contains method decides whether or not a point lies within the bounds of the graphic once drawn onto the canvas. This is used as part of the event system which loops over all contains methods to find a node or edge that a user is interacting with. The helpers

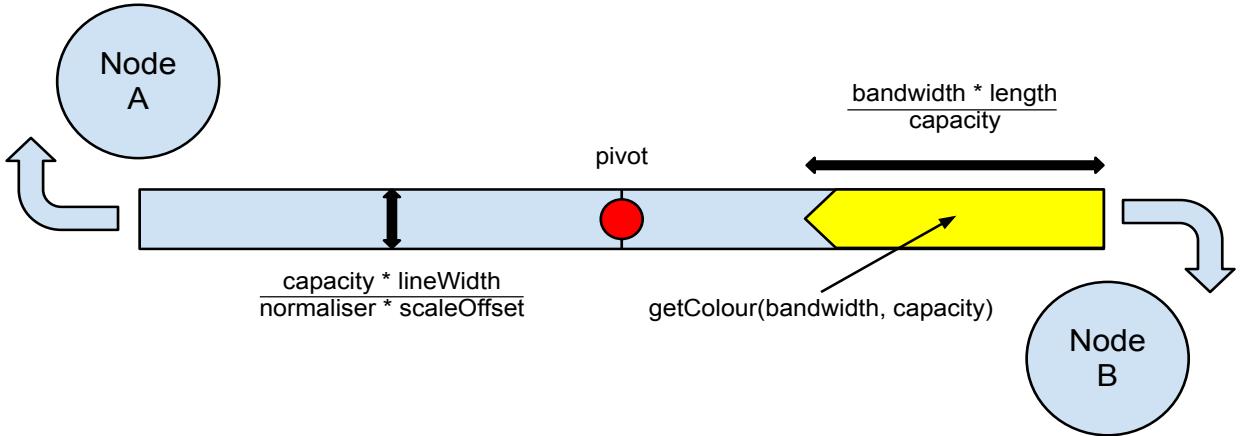


Figure 14: A diagram showing how map edges that include bandwidth data are implemented.

interface allows new edges and node types to be easily implemented.

NetMapJs implements several different node and edge types to cater for various functions. In particular, group nodes need to visually contain sub-nodes and edges need to consider the current level of zoom. As the user zooms into the network map, edge widths are scaled in proportion such that they always remain the same. This is required because in NetMapJs, quantitative variables may be encoded by the edge widths. If the scale for the widths changed as the user zoomed in, it would be difficult maintain an understanding of quantity. As more edges are added at greater zoom levels, multiple scales would have to coexist which would cause more confusion.

The main edge type that was implemented for NetMapJs was the bidirectional divided pipe as described in Section 4.2. Figure 14 shows how the edges are built using bandwidth as an example. A rectangle is drawn between two nodes with arrows facing in both directions. Sizes are calculated as shown, the canvas is rotated around a central pivot point and then the final pipe is drawn.

The height of the rectangle is calculated by normalising the capacity and factoring in the user defined line width and current scale offset. Capacity is the total bandwidth available over the link and the normaliser is a value that reduces the capacity value down to a smaller scale. The length of the arrows are calculated as the proportion of the bandwidth over the total capacity multiplied by the half of the rectangle's length.

5.3 Groups

Nodes can be assigned to a parent. If a node does not have a parent then it is considered to be one of the top level nodes. The top level is the highest level of the network map and is the starting point for grouping and depth algorithms. All of the nodes that do have a parent, become part of a group. Three methods handle the group functionality: BuildGroups, ComputeDimensions and ComputeLayouts.

BuildGroups recursively builds groups by assigning nodes and a depth level to a parent. A depth level is the distance, in groups, from the top to the parent's level. The function starts with the top level set of nodes and reduces it until all of the nodes have been assigned a group and depth level. The ComputeDimensions function loops through all of the groups and adjusts the node edge sizes. Node dimensions are scaled by a factor of the node's group depth such that deeper nodes appear smaller. Edges are assigned the line width of the parent which scales according to the current zoom level.

Groups can be assigned different layout algorithms for distributing nodes. The ComputeLayouts function first sets the root of a group if this has been specified in the initial JSON configuration. This is useful for algorithms that require a root to be specified such as the star layout described in Section 5.4. ComputeLayouts then loops through all of the groups and applies the specified layout algorithm to the group or the default if none has been chosen.

5.4 Layouts

Nodes need a way of being positioned on the screen. In computer networks there are many different layout styles that may make sense. For this reason, the layout system was designed to be easy to extend. Layout algorithms take an input, which is usually just a set of nodes and adjacencies, and output a set of positions for each node. NetMapJs defines a layout interface as an object that has initialise and compute methods. The layout is first initialised to set up any initial parameters and then the compute method is called to calculate node positions whenever they are required. The compute method may be called multiple times for algorithms that change over time such as the force directed layout. This system allows layout types to be changed at run time which could be used to give the user the ability to switch algorithms to achieve a different view.

NetMapJs currently has three layout types implemented: static, star and force directed. The static layout takes a set of node IDs that map to x and y coordinates and loads them into the nodes objects. The positions are sourced from the server that is keeping the state. The editor which is presented in Section 5.8 can be used to create these static coordinates and save them to the server.

The star layout algorithm places a given root node into the center of the screen and then places the rest of the nodes, evenly spaced, around the circumference of an invisible circle centered on the root. The radius of the circle is a specified fraction of the parent group's width. This layout is limited by the amount of space it uses due to the circular positioning of nodes. The space between the center of the circle and the circumference where all the nodes are located is used for edges. Also, the space between the circle circumference and the parent node's bounds is completely unused. If the parent node was not shaped like a circle then there would be gaps at the corners where the circumference can not reach. It is also only appropriate to use when the underlying network structure is star like.

The final layout algorithm, force directed, uses the Arbor library for maintaining the node positions. The purpose of force directed algorithms is to position nodes such that edges are similar lengths and there are as few crossings as possible. The initialise method of the layout interface creates the particle system and the compute method transfers the positions from Arbor to the node objects. This way, the compute method can be continuously called in order to give an animated effect as the nodes move toward their ideal positions.

5.5 Navigation

Navigation is implemented through the use of the Canvas translate and scale offset functions. The scale offset function performs a geometric zoom on the canvas which allows for full space-scale navigation in 2D [9]. In order to achieve semantic zooming where the information content changes based on the scaling, components on the network map must be matched to the levels of zoom. In NetMapJs, when a view is created it is assigned a set of scale offsets which define the points where information content should change. These offsets are computed with the depths of nodes to determine when they should be displayed as shown in Figure 15. Group nodes are automatically assigned a depth based on the number of parent nodes between them and the top level. Manual assignment of depth values is also possible.

A rendering function handles when and how each visualisation aspect should be displayed. Nodes, labels and edges that have a depth value that is less than or equal to the zoom offset are visible in the map. Nodes and edges that have a depth less than the offset are drawn with a lower alpha value. This creates the desired effect of fading out nodes above the current level. Edges have an additional condition that handles the case when edges join nodes of different groups. Those edges that extend from one group to another, deeper, group's node should only be partially displayed. That is, the edge should start from the displayed node and extend only to the edge of the group that contains the deeper node.

As described in Section 4.4, there are two navigation support features to assist the user in moving around the network map. The follow edges feature allows the user to click on an edge to be transported to the node on the other end. This is implemented by following a set of conditions. For a given edge that the user has clicked on, if both or none of the source and destination nodes are within the visual map space, then do nothing. If, however, one of the nodes is in the visual space then this becomes the source and the destination node is on the other end of the edge. The view port then follows a linear path towards the destination by translating the canvas until the node is centered on the screen. The zoom and center feature allows the view port to quickly focus on a node. This was implemented by calculating the translation difference required for centering and the scale offset necessary to zoom in

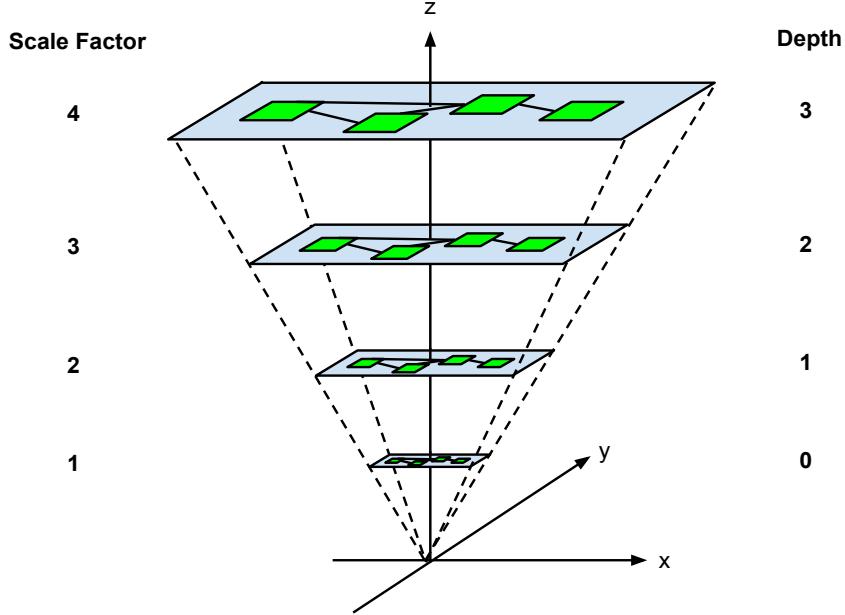


Figure 15: A space-scale diagram showing different levels of magnification being mapped to depth levels where the detail changes. (Adapted from Furnas and Bederson [9].)

on the destination node. A JavaScript interval is set to animate the view port into the new position by incrementing coordinates in each step.

5.6 Overviews

Overviews help the user to maintain overall context as they zoom and move around the network map. At higher levels of zoom, additional overviews are required to show each depth level of the network. An overview manager class handles the adding and removing of overviews when the main network map is navigated. An overview consists of a DOM container, a minimalistic network map instance and set of functions that handle the interaction with the main view. Overviews are added when the current depth level of the network map increases and removed when it decreases. The depth level is defined by the current level of zoom compared with the user defined scale offset values.

Overviews require a two way interaction with the main network map instance. If the user clicks on the overview to move the view port box to another position, then the main map needs to be translated accordingly. This translation difference is calculated by converting the overview coordinates of the mouse click into the main network maps geometric space and then subtracting the two. When the user navigates in the main view, the overview view port boxes need to move to reflect the change. Overviews listen for the redraw event to be fired by the main network map. When the event is triggered, the overview calculates new coordinates and dimensions for the view port box, and then redraws it.

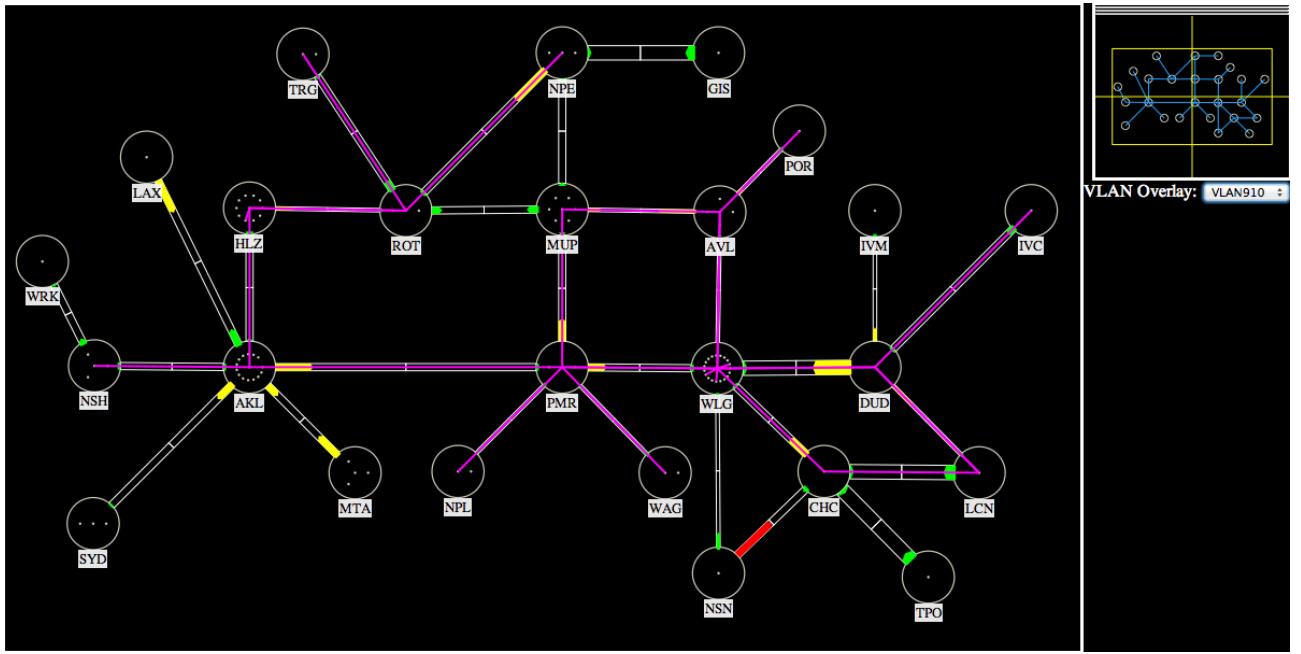


Figure 16: VLAN 910 for the KAREN network visualised using the overlay functionality.

5.7 Overlays

Overlays provide a way of adding additional visual features on top of the network map after it has been drawn. An overlay instance contains an update method that is called each time the overlay needs to be drawn. The method is passed a reference to the main visualisation object, the node and edge graph structure and the HTML canvas. These three references give the update method access to enough data to render overlays positioned relative to the network map elements. Overlays also contain methods to be started, stopped and redrawn.

A manager class is used to keep track of all of the overlays. It is assigned a network map object and uses it to pass the appropriate parameters to the overlay update methods. Overlay objects may be added or removed from the manager. Network map redraw events are bound in the same way as overviews. When this event is fired, the manager loops through all of its overlays and calls their update method.

One of the uses for the overlay functionality that was identified early in the project, and later requested for the KAREN map, is the displaying of virtual local area networks (VLANs). Different VLANs may take separate paths through a given network and may have their own set of performance metrics. An overlay that visualises VLAN flows separately for the KAREN network is shown in Figure 16. The purple lines shows the edges that the VLAN is attached to. Zooming into the map further would show the connections within sub groups. VLANs can be changed or turned off using the drop down box on the right hand side on the display.

5.8 Editor

A visual editor that allows map views to be defined in real time is particularly useful for getting visualisation parameters right. Ideally the editor should be able to perform minor alterations to node positions, apply layout algorithms to groups and set node and edge parameters. For example, different layout algorithms may need to be compared to find the best fit and some nodes may need minor subsequent position adjustments. The view, including all of the set configuration, needs to be able to be saved and loaded back later.

NetMapJs includes a basic editor that currently only supports nodes being moved around a grid. More work is required to produce a complete editing system. The editor was built using a combination of HTML, CSS, JavaScript and a special network map view. The view is an instance of the NetMapJs visualisation class that adds extra functionality to load and update node positions as they are clicked and dragged. When the view needs to be saved, the JSON data is taken from NetMapJs and sent to the server using AJAX. Previously saved views are loaded by reversing this process.

The editor was used to make the KAREN network map which is shown in Figure 17. An image showing one possible map design was copied by manually positioning nodes using the editor until a similar layout was achieved. The editor figure shows grids being used effectively for accurate node positioning and spacing. Zooming and panning functionality is also supported in the editor.

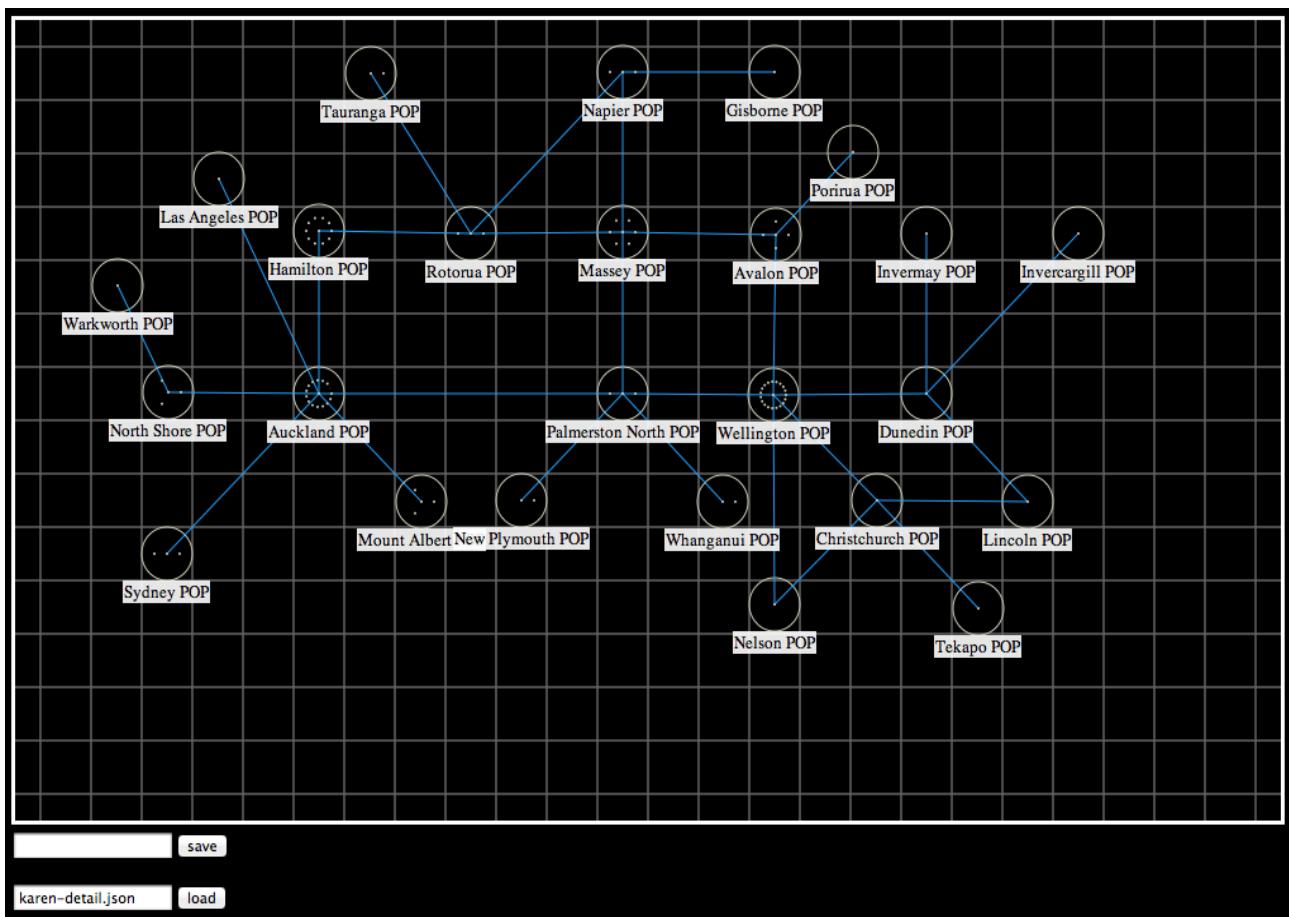


Figure 17: A screen shot of the KAREN network map loaded into the view editor.

6 Evaluation

An evaluation with network engineers from Rural Link and KAREN was performed to analyse the strengths and weaknesses of NetMapJs. The redesigns of their network maps were sent along with a set of questions (see Appendix C) to people in the respective networks.

Both networks found the edge graphics to be effective at displaying bandwidth data. Rural Link noted that the dynamic sizing of edge widths would be particularly interesting to test with live data. The example network maps produced in this project used static, randomly generated bandwidth values for drawing the edge graphics and as described earlier, NetMapJs supports polling this data and updating the map in real time. Live data updating the visualisation may raise other issues such as whether consistent large changes in bandwidth usage would distract users from other parts of the map.

The networks identified nodes as being effective at showing distant sub-network detail while still hiding information. Showing some limited detail of deeper sub-network levels indicates that the nodes exist and show at a higher level what group nodes have the most connected devices. The hiding of detail is effective at simplifying large network maps. Rural Link identified areas of the network map that could be converted to groups. Automatically deciding on what nodes to assign to a group can be difficult to get right and so it would be useful for NetMapJs to include an easy way of manually building groups. Selecting a set of nodes through user interaction in the map editor and then assigning that selection to a group would be a good way of implementing this.

The ability to view VLAN traffic separately through the use of overlays was another advantage identified by both of the networks. The shape of the VLAN as a subset of the overall network can be visualised using this feature. This shows that the overlay system is useful for supporting the development of add-ons to the network map. The VLAN overlay simply showed flows by drawing coloured lines over top of the normal edge graphics. There is much more potential for this overlay such as: visualising VLAN specific metrics, comparing overall link statistics with VLAN specific data and showing multiple VLAN flows at once using different colours.

The weaknesses of NetMapJs as identified by the networks were mostly user interface problems. In particular, more visual cues should be present to make some features more obvious and the styles used make aspects of the map hard to see. The visual cue, a tooltip, for following edges to the node on the other end only appear when the user hovers the mouse over the edge. A graphic that appears when following an edge without the user having to provide interaction would make this feature much more obvious.

Some of the colour choices on the map designs such as red on black make parts of the map hard to see. While more of the focus in this project has been placed on the development of new network map

display techniques, it is still important to make the design visually appealing. Users are not going to care about the advanced network map functionality if it does not look good. In NetMapJs all of the styles, colours, fonts, node and edge graphics and user interface layout are customisable. The visual styles were separated from the implementation to make it trivial for developers to customise the look of the map to suit the networks requirements.

Two issues with the user interaction around the zooming functionality were identified. When the user zooms in and out of the map, there should be hard limits imposed to stop them going to far in either direction. This would support the user to get an idea of the size of the network without having to guess based on the fact that no more detail is being added from zooming in or there are no more nodes off the screen when zooming out. The zooming out limit could be based on the minimum and maximum coordinates of the furthest nodes in each direction on the screen. Zooming in could be restricted to the deepest sub-network in the map.

The other issue with zooming is that it is based on the center of the screen not the mouse position. So when the user wants to zoom in on a particular area of the network map, they must perform suitable translations of the view to center it around that area before zooming. More panning may be required if the view port is not exactly centered over the area of interest. A much more natural way of doing this is to zoom towards the mouse position in an attempt to center the view port around that. This way, the panning and zooming are easily achieved together using the mouse pointer position.

A comparison of NetMapJs to the networks existing network maps found it to be much more interactive, even more so than the new Icinga network management system [14]. NetMapJs was able to visualise the redundant loops in Rural Link's network where the Nagios map could not. Static layouts and the visualisation of networking performance data are also improvements of NetMapJs over Nagios. The PHP Network Weathermap is a more complete application that includes support for common database integration and animation of static images that show how bandwidth is changing over time. However, the dynamic nature of NetMapJs which allows for improvements such as easy overlay integration, semantic zooming and live update support will make it an improvement over this tool when the issues with the user interface have been addressed.

Both of the networks that evaluated NetMapJs would consider using it as a replacement to their current network map and mentioned that it would make a good upgrade. KAREN noted that NetMapJs had provoked some thought in their team. Overall, this feedback shows that this project had been effective at developing improvements to computer network maps.

7 Conclusions and Future Work

The goal of this project was to produce an improved computer network map display. This was achieved by identifying weaknesses in existing implementations and looking at ways of addressing them. A web based application called NetMapJs was developed to generate and control interactive network map displays. The main features of NetMapJs developed for this project are:

- Data adapters that separate raw data from the actual visualisation.
- Semantic zooming and panning combined with sub network hierarchy for improved scalability.
- Automatic layout algorithms that can be applied to sub network groups.
- Stacked overview boxes that help users maintain a sense of positioning as they navigate the map.
- An overlay system that provides a generic way for developers to add in additional graphics over the network map.
- An editor view that allows users to statically position nodes.

Through the use of these features, NetMapJs was able to address some of the limitations of existing network map implementations as described in Section 1. The outcomes of this project in terms of network map improvements are:

- Rich user interactivity allowing the user significant control over the visualisation.
- Scalable network map design that allows large networks to be displayed without hitting performance limits.
- A web browser based display using HTML5 specification that does not require specialised software.
- Informative edge graphics incorporated into edges that display network metrics using sound information visualisation theory.

NetMapJs was tested using real topology data sourced from the Rural Link and KAREN networks. An evaluation using these two networks was successful and pointed out the strengths and weaknesses of NetMapJs. Engineers from both of the networks indicated that this network map would be an improvement over their current tool in use and that they would consider using it once the software was polished and their concerns addressed. In particular, with an improved user interface, a more complete set of layout algorithms and a completed client/server side model implementation, NetMapJs will be ready for live deployment in real world systems.

There are several possible paths to take for future work resulting from this project. Semantic zoom could be investigated further to explore possible uses with respect to network maps. Early on in this project, the ability to relate different network layers together through the use of zoom was considered but little progress made. Very large networks, such as an entire autonomous system, may require that zooming splits up data requests and only displays currently visible depth levels.

This project only considered two automatic layout algorithms out of the many that are found in literature [5]. Including more layout algorithms, particularly those that are known to be effective for computer network visualisation, would allow a more diverse set of networks to use the tool. One notable feature included in some other network mapping tools is the inclusion of graphs of relevant metrics. For example, graphs of bandwidth over time are useful to help network engineers identify when anomalies occur. These graphs could be included as part of the visualisation itself through the use of user interaction triggered events.

NetMapJs includes a rich design for displaying network metrics between two nodes as part of the actual edge graphics. A survey of node and edge designs for computer networks would support the development of further network map displays. In this project, NetMapJs was developed as a stand alone application. Incorporating the tool into a network management system such as Nagios [21] or OpenNMS [22] would be a good extension to allow NetMapJs to be used with a minimal installation and setup overhead.

References

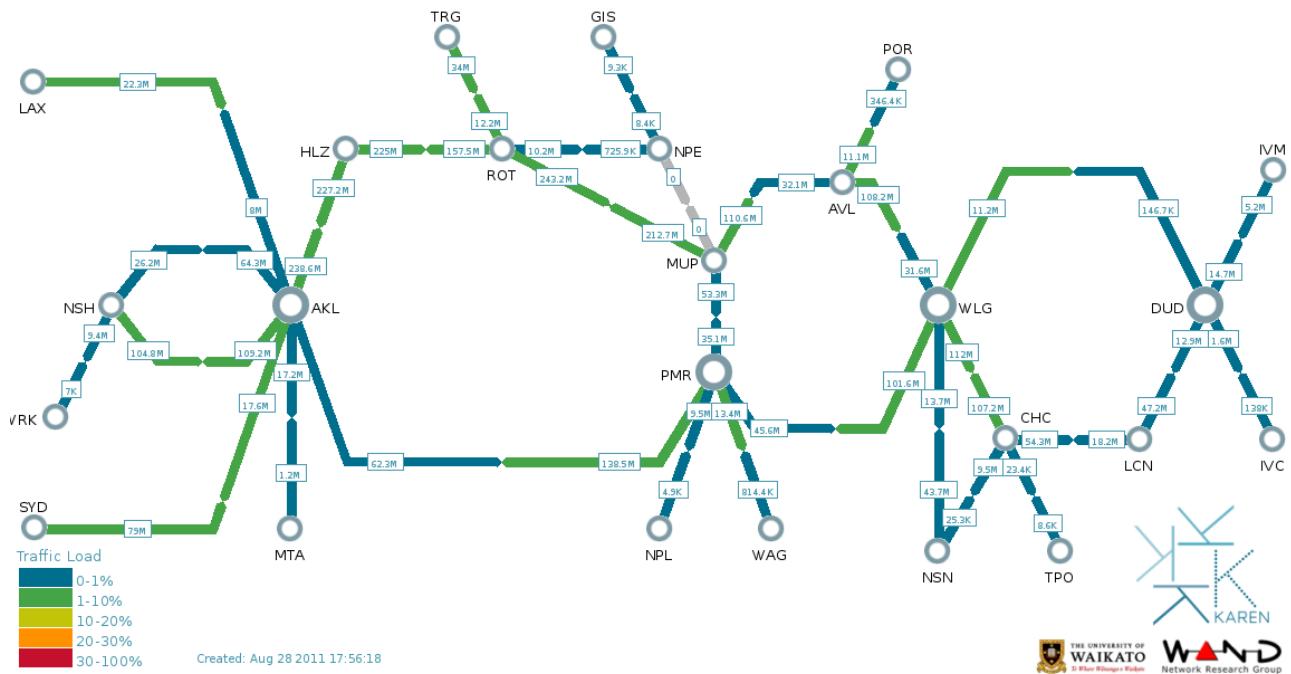
- [1] R. A. Becker, S. G. Eick, E. O. Miller, and A. R. Wilks. Dynamic graphics for network visualization. pages 93–96,. 1st conference on Visualization '90.
- [2] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):16–28, March 1995.
- [3] N. G. Belmonte. Javascript infovis toolkit. <http://thejit.org/>. Accessed October 8, 2011.
- [4] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). *RFC 4627*, 2006.
- [5] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, October 1994.
- [6] S. T. Eick. Aspects of network visualization. *IEEE Computer Graphics and Applications*, 16(2):69–72, March 1996.
- [7] Stephen G. Eick, M. Andrew Eick, Jesse Fugitt, Brian Horst, Maxim Khailo, and Russell A. Lankenau. Thin Client Visualization. In *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on*, pages 51–58, October 2007.
- [8] Jon Ferraiolo, Jun Fujisawa, and Dean Jackson. Scalable vector graphics (SVG) 1.1 specification. Technical report, World Wide Web Consortium, 2003.
- [9] George W. Furnas and Benjamin B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95*, pages 234–241, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [10] Andre Girard. *Routing and Dimensioning in Circuit-Switched Networks*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990.
- [11] I. Herman, G. Melancon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000.
- [12] I. Hickson. The canvas element html standard. <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>. Accessed October 8, 2011.

- [13] I. Hickson. HTML5 - A vocabulary and associated APIs for HTML and XHTML. <http://dev.w3.org/html5/spec/>. Accessed October 19, 2011.
- [14] Icinga. Open Source Monitoring. <https://www.icinga.org/>. Accessed October 12, 2011.
- [15] Mikael Jern. Information Visualization on the Web. *Information Visualisation, International Conference on*, 0:2+, 1998.
- [16] Donald W. Johnson and T. Jankun J. Kelly. A scalability study of web-native information visualization. In *Proceedings of graphics interface 2008*, GI '08, pages 163–168, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.
- [17] H. Jones. Network weathermap. <http://network-weathermap.com/>. Accessed October 7, 2011.
- [18] Karen. Kiwi advanced research and education network. <http://karen.net.nz/>. Accessed October 7, 2011.
- [19] W. Lai and P. Eades. Routing Drawings in Diagram Displays. In *Proceedings of the Third Asian Pacific Computer and Human Interaction*, Washington, DC, USA, 1998. IEEE Computer Society.
- [20] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [21] Nagios. Nagios - The Industry Standard in IT Infrastructure Monitoring. <http://nagios.org>. Accessed October 7, 2011.
- [22] OpenNMS. The OpenNMS project. <http://www.opennms.org/>. Accessed October 10, 2011.
- [23] Richard Z. Paul, Paul Coddington, and Ken Hawick. Computer Network Visualisation. Technical report, Department of Computer Science, Adelaide University, South Australia, 2000.
- [24] L. D. Paulson. Building rich web applications with Ajax. *Computer*, 38(10):14–17, October 2005.
- [25] Ken Perlin and David Fox. Pad: an alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 57–64, New York, NY, USA, 1993. ACM.
- [26] J. Resig. jQuery: The Write Less, Do More, JavaScript Library. <http://jquery.com/>. Accessed October 8, 2011.

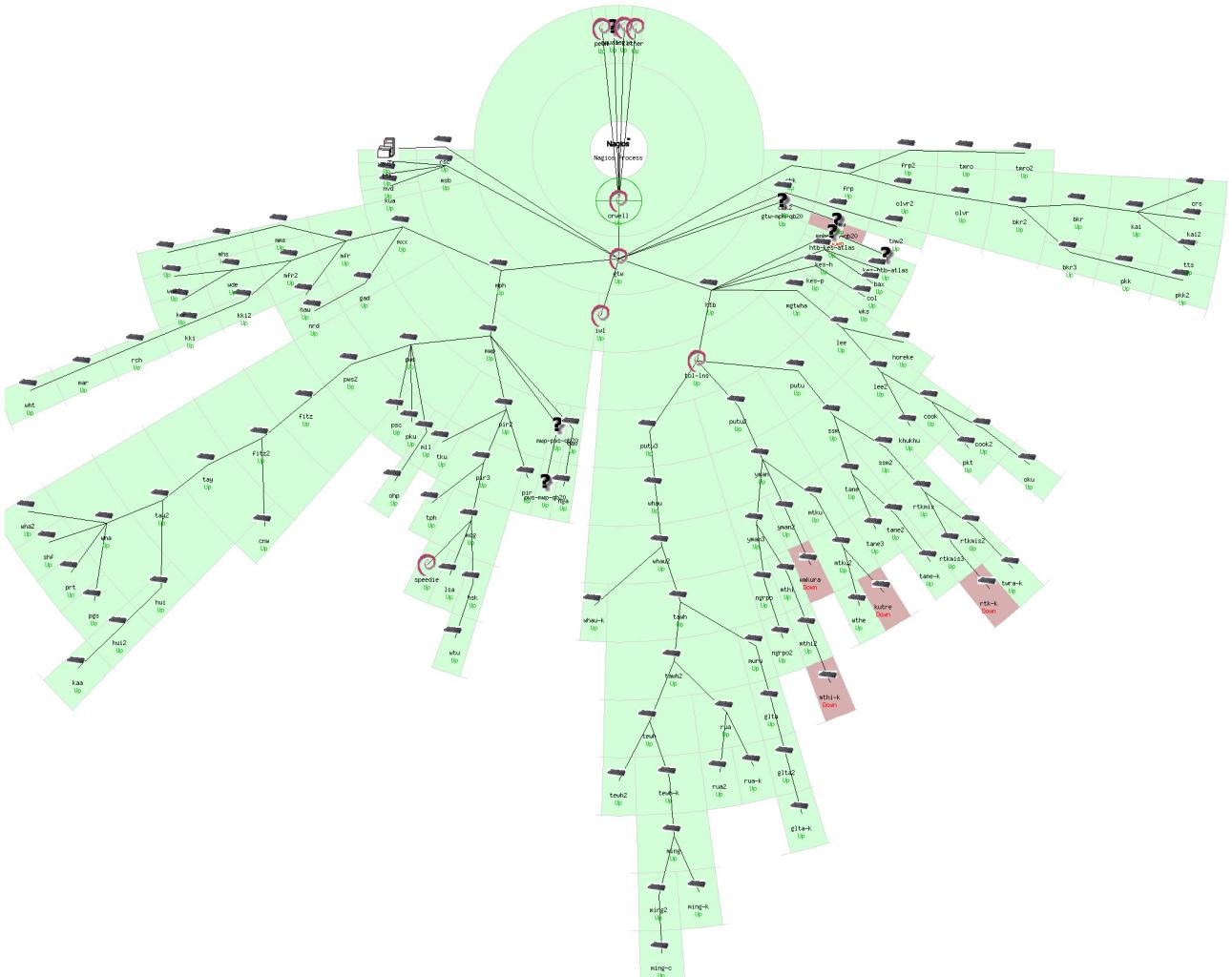
- [27] R. M. Rohrer and E. Swing. Web-based information visualization. *IEEE Computer Graphics and Applications*, 17(4):52–59, July 1997.
- [28] Robert Spence. *Information Visualization: Design for Interaction*. Pearson Education, 2007.
- [29] C. Swinehart. A graph visualization library using web workers and jquery. <http://arborjs.org/>. Accessed October 8, 2011.
- [30] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, August 2007.
- [31] Jason Wood, Ken Brodlie, Jungwook Seo, David Duke, and Jeremy Walton. A Web Services Architecture for Visualization. pages 1–7, December 2008.
- [32] Zenoss. Open Source Network Monitoring and Systems Management. <http://zenoss.org/>. Accessed October 10, 2011.

Appendices

A KAREN PHP Network Weathermap



B Rural Link Nagios Network Map



C Evaluation Question Set

1. What do you think are the strengths of NetMapJs?
2. What are the weaknesses of NetMapJs?
3. Would this map be one that you would consider using in your network? Why? / Why not?
4. How does NetMapJs compare with your current network map?
5. Do you have any other feedback?