Visualizing Network Agents

Ken Deeter

April 29, 2003

1 Introduction

Information visualization systems attempt to transform abstract data into a visual representation to aid in its comprehension and management. In the case of networks, one can frequently be faced with abstract network topologies, and behaviour in the network that does no intuitively map to any visual representations. Network developers often rely upon the convention of a node-link graph to depict topologies, and there has been much work attempting to take advantage of this convention to automatically visualize these complex topologies.

My goal in this project was to apply various information visualization techniques to the area of mobile network agent based computing.

When developing a system based on the mobile agent model, it is often the case that the behaviour of the system is difficult to comprehend because of its highly parallel nature.

1.1 Target Task

The system presented in this paper, temporarily named WaveViz, is intended to be used by developer of algorithms based on the agent paradigm. The intent is to allow developer to visually verify the behaviour of their system, over a reasonably large network topology.

Several specific requirements were identified:

- Because an agent based system or algorithm is inherently highly parallel, parallel events in the network must be able to be seen as happening in parallel. Although this may seem trivial, it imposes very specific requirements on the implementation.
- When operating on a large network topology, a user needs to be able to focus in on a specific area of a network. For example, if it is known a priori, that an algorithm fails at specific nodes, it should be possible to easily "zoom in" on to those nodes.
- Some network topology browsing should be available. When dealing with a system that runs on a network, it is often necessary to understand the underlying topology. The system should support some method of understanding complicated topology.

1.2 Data

The input data to WaveViz is a log file of network events. The syntax of this log file was designed to be as flexible as possible by supporting arbitrary network topologies, as well as dynamic topology changes (non-static topology). The intent was to allow WaveViz to be used by any relevant agent based system. Each system only need to be modified slightly to log data in a specific format. None

of the data required is anything a agent based system would not already know about itself, confining the modifications of a system to simply the capture of specific events, and not the discovery of state.

1.3 Assumptions

In the design of the system, several assumptions are made regarding the type of data input and its scale.

- This system avoids the traditional graph layout problem by assuming that the input data provides 2D locations for each node. Although this limits the system's usefulness to some extent, the ideas that it demonstrates can be applied to systems that have automatic graph layout facilities as well.

  In practice, location data is often available and very relevant in visualizing topologies. Typically, hosts on a network have some physical location in space (one which may even change as time goes by). Automatic graph layout can be useful in understanding the *abstract* structure of a network. In the scope of this project however, it was a reasonable to assume that we only care to see a more *physically-based* representation of a topology.

- WaveViz is also not designed to deal with very dense networks (i.e. high edge/node ratio). Networks in the real world tend to be sparse, with links often connecting only the nearest of neighbours.

2 Related Work

In spirit, this project is in many ways similar to the super-scalar processor visualization described in Stolte et al. [5]. Both attempt to use visualizations to help users understand inherently parallel time-varying data.

WaveViz also borrows many ideas traditional zooming interfaces such as those described in Furnas et al. [2] and Bederson et al. [1]. Although no animated zooms were implemented in WaveViz, the simple idea of interactively focusing in on a subset of data is an integral part of this system.

The screen distortion features in WaveViz are inspired by works such as Munzner[4], Lampling et al, and the variety of methods described in Leung et al [3].

3 Approach

3.1 Interaction Model

In designing the system, a specific model of interaction with the user was kept in mind. A development cycle similar to the following was envisioned:

1. Edit/compile agent program/algorithm
2. Generate log
3. Feed log into WaveViz
4. replay in WaveViz to confirm correct behaviour
5. repeat

In this model, a user (developer) knows what he/she needs to see when using the visualization system. Thus, there is no need for the system to support a exploratory interaction model. It is assumed that the user will have some expectation of what the displayed behaviour of the system will be.

3.2 Zooming

The emphasis, then, is on allowing the user to concentrate on specific aspects of the behaviour. To allow this controlled concentration, a modified style of

zooming is introduced. The zooming controls in the visualization are not arbitrary. The user cannot zoom in or out as far as he/she wants. Any notion of zoom is defined with respect to elements in the network.

In WaveViz, this idea is most well illustrated by a method of a zooming that "fits" elements of interest onto the available window space. The system, by default, attempts to fit all nodes of the network onto the screen. The user, however, can easily select any subset of nodes, and the zoom automatically readjusts to fill the screen with these subsets of nodes.

There are two points of interest with this type of zooming. First, the system can guarantee that most of the screen space available will be used. As pixels are limited resource, this type of zooming was designed to always take advantage of the pixels available. Secondly, because the zoom is always defined in terms of elements of interest, a user cannot get "lost."

3.3 Screen Distortion

Even with this zooming model, it is often the case where even a reduced set of on-screen elements can be displayed too closely together. In WaveViz, if the position of two nodes are specified very closely to each other in the input, then no amount of zooming alone will automatically un-clutter these two nodes (unless we zoom in on a subset containing only those two nodes). Because of this, a few different methods of screen distortion are added to allow the user to "unclutter" the displayed topology.

These distortion mechanisms are designed to controllably distort the positions of on-screen elements, so that a user can visually expand a specific area within a particular zoom. Care is taken to ensure that a the subset of nodes that are selected for the current zoom are always on screen, even with distortion applied. These distortions provide a focus+context type interface which allows groups of nearby nodes to be allocated more screen space if the so wishes.

3.4 Faded display

The zooming is implemented by computing a bounding box on the selected nodes of interest (see section 4 for more detail). The region in model space described by this bounding box is then scaled to the available rectangular screen area. When drawing the region of interest, however, it is possible that nodes that were not selected to be zoomed in on, may still fall inside this bounding box. As these nodes (and links attached to them) may still constitute important contextual information, they are drawn in a faded colour, so that they are still visible, while not drawing attention from the main elements of interest.

3.5 Frame rate

During any times where user input causes a change of visual state, the system attempts to display a reasonably smooth animation. Certain screen elements may be not drawn during animation phases, maintaining frame rate, frame-to-frame coherence, and ultimately, user comprehension.

3.6 Subset Selection

When selecting a subset of nodes to zoomed in upon, it can be difficult to see node connectivity, as all nodes in the network are displayed simultaneously. Because of this, a very simple method of showing network topology is implemented while a selection is taking place. Specifically, while each selected node is highlighted with the selection colour, every non-selected node adjacent to a selected node is highlighted with a secondary highlight colour. A user can immediately tell which nodes are adjacent to the nodes that are currently selected. This technique aids

the user in selecting a connected subset of nodes, which is often analogous to selecting a subnetwork.

4 Implementation

4.1 Log file format

The log file which serves as input to WaveViz has a very simple format with one line per entry. A network is define by a collection of three fundamentally entities: nodes, links, and agents. In its current version, approximately one dozen simple operations on these entities are supported, including the creation/deletion of nodes/links/agents, the movement of agents, and state changes for nodes/links/agents.

"States" for various entities are define by special headers in the log file. A identification string is mapped to visual parameters for the various entities. For example, one might define the "supernode" state as being represented by a node with red colour and large radius. The number of such states is not bounded, and thus can be used to represent an arbitrary number of application specific states.

Each line contains one event in the network. Each event is prefixed by two timestamps – a start time and an end time. Events that happen instantaneously, such as the creation/deletion of nodes, are assumed to have the same value for both the start time and the end time. Events that happen over a period of time, are assumed to have different start and end times. By having both start and end times in each entry, we can easily record events happening in parallel.

Log entries in the log file do not necessarily need to happen in order. All entries are sorted by their start time prior to replay. This flexibility allows for more convenient log collection mechanisms in the actual systems that are to be visualized, as collecting logs of multiple parallel distributed events can be a challenge.

4.2 Visualization Engine

The engine was written in Java2D (and the application in Java). Java2D was chosen as it provided a convenient library for drawing shapes. In retrospect, however, implementation with a more efficient library such as OpenGL would have allowed more complex behaviour.

The engine is organized into a pipeline. The pipeline has the following stages:

1. Zoom: this stage performs a transformation of node positions from a virtual model space (as specified in the log input) to screen space, depending on the current zoom bounding box. This bounding box either is set to contain all the nodes in the graph (and dynamically adjusts when new nodes are created or destroyed) or a selected subset of nodes.

   This phase can also determine the visibility or "fade" of nodes. This is mainly done by testing whether each element falls inside the current bounding box.

2. Distortion: The distortion phase performs a mapping from screen space coordinates to screen space coordinates, depending on the current distortion parameters.

3. Render: The final phase renders the network elements onto the screen with the parameters determined by the previous phases. The colours that are used in for various elements are calculated in this phase from higher-level parameters (such as "fade") determined in the previous phases.

Both the Zoom phase and the Distortion phase are performed by modularized classes separate from the main application, allowing for easy replacement and expirementation. In developing the system, several

ideas were tested, but a simple design was chosen due to Java2D's performance limitations.

5 Results

Figures on the following pages illustrate the various aspects of the system previously described. The performance limit for the system appears to be around 500 nodes, although no formal performance testing has been done.

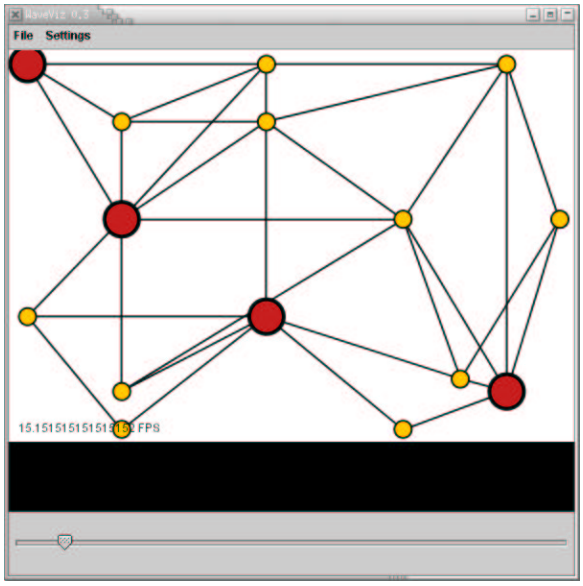

Figure 1: A screenshot of the system showing a simple network of 15 nodes with 4 nodes having differentiated states.

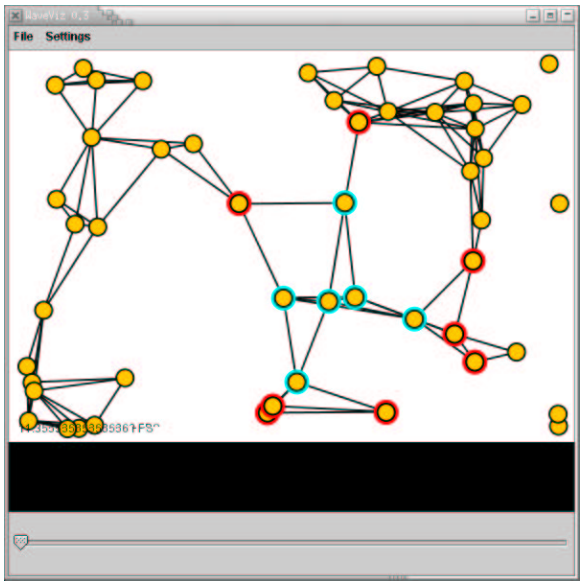

Figure 2: A screenshot of the system showing a network of 50 nodes, with the user in the process of selecting several nodes. Selected nodes are shown with the cyan hilight, and nodes adjacent to selected nodes are shown with a red hilight.

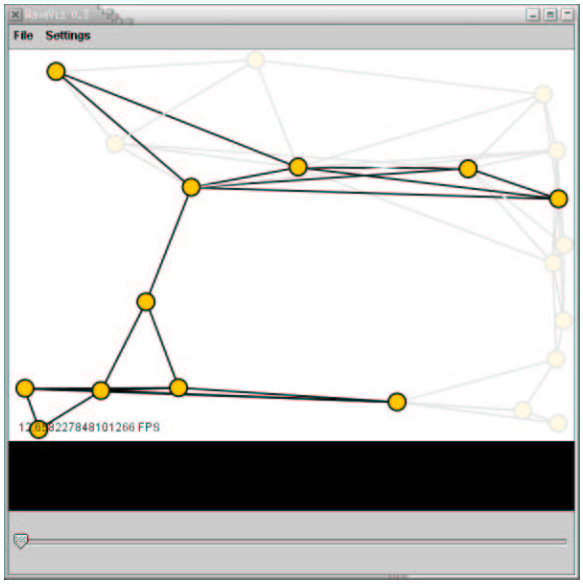

Figure 3: A shot of a subset of nodes (from the 25 nodes in the previous figure), shown with a zoom. Faded nodes show location of nodes that fall within the zoom's bounding box, yet have not been explicitly selected.

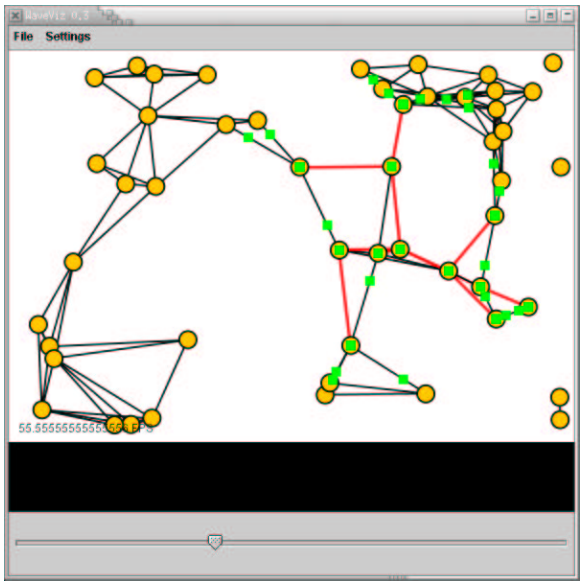

Figure 4: A shot of a multicast backbone creation algorithm in process. Agents are shown with green boxes. Red links depict links chosen to be part of the multicast tree.

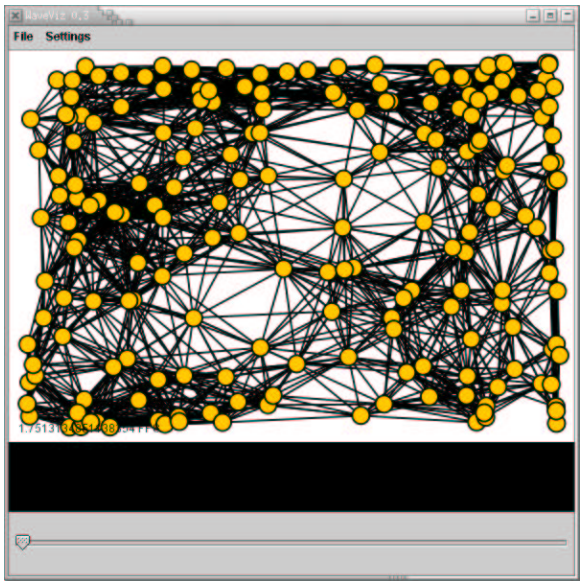

Figure 5: A shot of a 200 node graph.

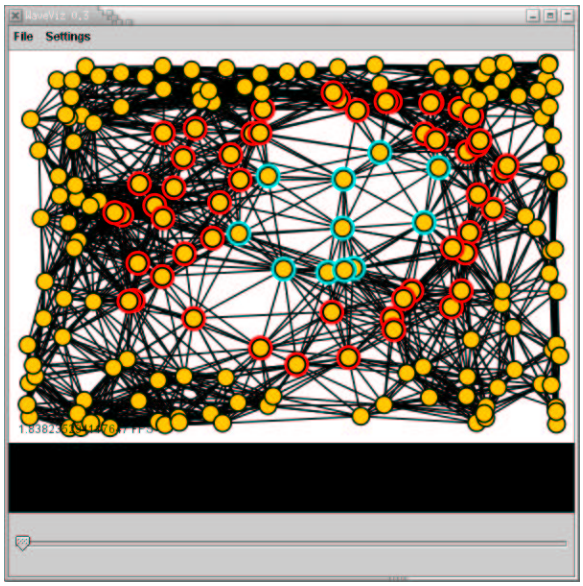

Figure 6: A shot of the same 200 node graph during a selection process.

6 Discussion

6.1 Strengths

The system in its current state already provides a useful tool for developers. In demonstrating this tool to several networking grad students, they have expressed significant interest in pursuing further use. The ability to see parallel events in the network is useful, but the ability to manage a larger number of nodes in an arbitrary configuration is the differentiating factor of this design.

6.2 Weaknesses

The major weakness in the system is, in my personal opinion, its incompleteness. In concept, I believe the ideas in this system have much potential. However, due to various reasons (lack of Java2D performance, lack of time), the implementation could not fully realize all the ideas in a complete manner.

Many issues exist with the current system which need to be addressed. Some as simple as additional UI requirements, some more complex as modifying the rendering code.

There are many bugs and problems related to layering and drawing order. Some nodes appear above other nodes in an inconsistent manner. Faded nodes may be drawn over unfaded ones. Hilighted links may be obscured by ordinary ones. All these problems are due to the simpleness (or ignorance) of the rendering routine. In many cases the rendering engine was left unfixed to avoid incurring any further rendering costs.

In addition to more intelligence during the rendering process, greater user control in the rendering output would be desirable. For example, the ability to emphasize certain links (such as the multicast tree links in the multicast example) would make the visual display much more comprehensible, when many overlapping links are being displayed.

The distortion methods employed could also be improved. Although they allow for somewhat useful distortion, they leave much to be desired. The methods used by WaveViz are consistently inadequate when a user is interested in multiple areas of a zoomed subset. A multi-focus lens-like approach may be able to solve this problem.

Zooming could be improved by including parameters such as the amount of available screen space, into the layout algorithm. The current bounding box-based method does not produce good results in many cases, such as when relatively few nodes are selected for a zoom. In this case, distance between close nodes can be greatly increased, reducing the recognizability of links between them.

References

[1] Ben Bederson and James D Hollan. Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proceedings UIST*, 1994.

[2] George Furnas and Ben Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings SIGCHI*, 1995.

[3] Y.K. Leung and M.D. Apperly. A review and taxonomy of distortion-oriented presentation techniques, June 1994.

[4] Tamara Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *Proceedings Information Visualization*, 1997.

[5] Chris Stolte, Robert Bosch, Pat Hanrahan, and Mendel Rosenblum. Visualizing application behavior on superscalar processors. In *Proceedings Information Visualization*, 1999.