# NView: A Visual Framework for Network Tool Integration *

David G. Thaler and Chinya V. Ravishankar
Electrical Engineering and Computer Science Department
The University of Michigan, Ann Arbor, Michigan 48109-2122
thalerd@eecs.umich.edu ravi@eecs.umich.edu

## Abstract

*Monitoring and managing distributed applications and networks present a number of problems. Tracking down the source of connection problems over a large area, such as the Internet, can be time-consuming and confusing, as one must deal with many small utilities. In this paper, we present NView, a framework program for integrating network tools into a uniform visual interface with a minimum of effort. We also discuss an example implementation which has been applied to the Internet's Multicast Backbone. This easily customized tool allows various "what if" questions to be asked, and results to be displayed in different ways. NView can be used both off-line as a design tool, and on-line as a network monitoring tool.*

## 1 Introduction

While hardware capabilities are improving rapidly in the computing and networking domains, our abilities to manage and control large networks are still inadequate. Future networks must make efficient use of new technologies such as multicasting, and support high bandwidth applications such as voice and video transmission.

Currently, network management is difficult. Many tools exist, but are typically used in an ad-hoc manner, with little integration. New tools have been introduced for technologies such as multicasting. Since new capabilities are added incrementally, new tools are usually designed by different people at different times, resulting in a heterogeneous collection of utilities. The number of utilities needed to effectively manage the networks of the future will likely continue to increase. Protocols such as SNMP (Simple Network Management Protocol)[1, 2, 3], are designed to unify network management, but total support for SNMP is still far off.

The continued evolution of applications such as Gopher, the World Wide Web, and the MBone (Internet Multicast Backbone) have created unanticipated loads which require special management utilities. While SNMP helps manage the networks themselves, it requires kernel support and is not designed to manage distributed applications without kernel modifications or support from a multiplexing protocol such as SMUX[4] or DPI[5], which do not have widespread usage. Again, the result is a piecemeal conglomeration of non-uniform utilities. A uniform interface is desirable to address the problem of confusion. A software framework, similar to a workbench or development suite, into which existing tools may be integrated can provide a simple yet powerful visual user interface for network debugging and monitoring.

We divide network management into two sub-problems: managing the topology of the network itself (i.e. the network layer and below), and managing distributed applications (i.e. the transport layer and above). The first ensures the existence of communication links, while the second deals with the actual use of links. For our purposes, we will refer to these simply as the "network" layer of connections, and the "application" layer of connections.

Our goal is to present a framework which helps to solve both problems with an interactive, on-line tool. Off-line debugging is not sufficient for general problems, since out-of-date information is often useless. We also want an integrated visual interface, rather than a simple text dump, to display current information. The ability to comprehend the status of the network easily is often a key to effective management. We also require that response time be fast, that the user be able to issue commands without waiting for unrelated actions to complete, and that the system avoid duplicate effort wherever possible.

If these goals are met, it should be possible to prevent many serious network failures, such as the one discussed in section 5.1 causing the entire MBbone to go out of operation for over 24 hours. It should also be possible to reduce the amount of time needed to solve problems with existing tools.

The remainder of this paper will be organized as follows. Section 2 discusses a number of existing tools, section 3 describes the overall structure of NView, section 4 covers some features of NView, section 5 then presents an example application of NView to the MBone, and section 6 covers conclusions and the future.

## 2 Existing Tools

There are many existing tools to facilitate different aspects of network management. Among the relevant ones are Unix utilities, TCP/IP management protocols such as SNMP, and other network managers. Such tools are individually useful, but they must be used in conjunction with each other to

*This work was supported in part by National Science Foundation Grant NCR-9417032.

0-7803-2492-7/95 $4.00 © 1995 IEEE

achieve optimum benefit.

Unix utilities such as `ping` and `traceroute`[6] are text-based and deal only with the lower layers of the network. They are relevant to our first problem of describing the network topology and status. However, they provide no visual interface, require the user to manually enter commands to retrieve each piece of information, and allow no easy method for accumulating data over time or integrating results with other tools. Therefore, on a large scale, using small tools one by one is not practical. Other utilities, such as `graphed`[1] exist for generating aesthetically pleasing renditions of graphs, but are not designed for integration with network tools.

SNMP[1, 2, 3], a network management protocol in the TCP/IP suite, provides methods for retrieving arbitrary information from hosts on an IP network. However, many nodes on the Internet do not respond to SNMP queries, so that any tool which requires SNMP conformance will be limited to the subset of the Internet which supports SNMP. This is currently unacceptable, although it can be very advantageous when nodes do support SNMP.

Lastly, a number of commercial network management tools exist with which we may draw comparisons. We will briefly look at two examples. IBM's AIX NetView/6000[7] does provide SNMP capability and on-line monitoring with a visual interface. It employs a two-level structure, where subtrees of nodes extend from a single, top-level "backbone" of networks. It also allows custom features and applications for performance monitoring. However, it requires SNMP capabilities on all nodes, runs only under AIX, and requires manual placement of nodes on displays.

IBM's INTREPID[8] also provides a visual framework, but is intended for network topology design. It also utilizes a two-level structure of nodes similar to that of NetView/6000. INTREPID allows the entire network to be redesigned for optimal topology and permits manual design modifications. However, in the Internet, the presence of a single backbone is not a safe assumption. Finally, INTREPID requires the user to prepare an input file, and cannot discover initial information itself.

Table 1 summarizes the features available in other systems we have discussed and shows how NView compares with them. These features will be discussed in more detail later.

| Feature | NetView /6000 | INTREPID | mrdebug[2] | NView |
|---|---|---|---|---|
| Graphical display | Y | Y | - | Y |
| Net decomposition | P | P | - | Y |
| Automatic layouts | P | P | - | Y |
| Tcl/Tk standard | - | - | - | Y |
| Design capabilities | - | Y | Y | Y |
| Maintains database | Y | Y | P | Y |
| On-line updates | Y | - | - | Y |
| Asynchronous traps | Y | - | - | Y |
| Unix commands | P | - | - | Y |
| Works with SNMP | Y | Y | - | Y |
| Works without SNMP | - | - | Y | Y |
| Portable among OS's | - | - | Y | Y |
| Freely available | - | - | Y | Y |

Key: Y = Full Support, P = Partial Support, - = No Support

Table 1: Comparison of Features

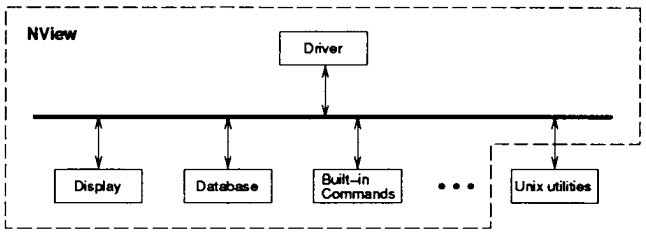


Figure 1: Internal Structure of NView

## 3 Structure of NView

NView is a framework for combining network tools into a single uniform interface which may be used for network monitoring and debugging. Figure 1 shows the architectural schematic of the framework.

The basic internal structure is that of a bus to facilitate communication between modules. A driver module acts as the parser and bus controller, while other modules provide a display, database, and other commands and utilities. This architecture allows the power and flexibility of interaction and cooperation between modules.

Internally, NView uses `Tcl` to implement the core of its communication between modules. This provides flexibility, standardization, and the ability to easily communicate with other `Tcl`-based utilities. Some commands are interpreted directly by the driver module. Other commands are passed to the appropriate module, which is free to issue additional `Tcl` commands or C function calls to other modules.

The display module uses `Tk` to generate an X interface when the `DISPLAY` environment variable is set, and Curses to generate a visually compatible full-screen interface otherwise.

The database module maintains a database of the network topology as currently known, and holds information on each node such as hostname, IP addresses, current status, and SNMP version. Other information on links such as current status and link type is also maintained. Keeping such information in the database between runs obviates the need to repeatedly re-acquire common information such as hostnames which may be time-consuming.

`Tcl` provides a language for communication between modules, but the NView driver extends this capability by adding the concept of *output processors*, discussed in more detail in Section 3.1.2. To this end, the driver maintains an internal *socket registry*, which maps sockets to output processor routines. NView will accept input on any registered socket.

The NView driver also contains a mapping between command names and associated routines, as well as a mapping

[1] Available as ftp://forwiss.uni-passau.de/pub/local/graphed/graphed-4.0.8-alpha.tar.Z

[2] `mrdebug` is a tool specific to the MBone (see section 5.2)

between "hot keys" and command macros. This latter facility is provided by Tk for X windows, but a separate mapping is required when running in Curses mode. Some examples of mappings are shown in Figure 2.

Tcl also provides the ability to define variables and expand them in commands. NView exploits this feature to enable the user to quickly access common functions with a single keystroke, while having maximum flexibility to also enter full command strings, or even redefine the hot key bindings.

```
Curses:
   map > nv_rotate 5
   map f nv_exec finger @$hostname
   map p nv_start ping_hdlr ping -n $ip 64 1

Tk:
   bind .c <KeyPress-greater> {nv_rotate 5}
   bind .c f {nv_exec finger @$hostname}
   bind .c p {nv_start ping_hlr ping -n $ip 64 1}
```

Figure 2: Sample macros

Other built-in commands and external utilities provide access to common features such as automatic graph layout, node update, and network decomposition algorithms. These will be covered in more detail in Section 4.

## 3.1 Tool Integration

There are three levels at which another tool can be integrated into NView: as a built-in command, using output interpretation, and without output interpretation.

### 3.1.1 Level 1: Integration as a Built-In Command

In the first and strongest level, the utility is compiled directly into the framework itself, where it becomes a built-in command. This type of integration works best with algorithms which are used often, such as ping, and those which manipulate large portions of the network database, such as graph layout algorithms. An obvious additional requirement is that the source for the utility be available, though it can be written in either C or Tcl.

This level requires the most work of the three levels, since it may require porting the utility for use with NView. However, once a standard interface is added, one simply adds the command name and function to execute to the internal command list in order to complete the integration.

### 3.1.2 Level 2: Integration Using Output Interpretation

In the second, and medium level of integration, the utility remains a separate process, so that source code for the utility is not needed. However, an output processor must be compiled into NView, or written in Tcl. An output processor is

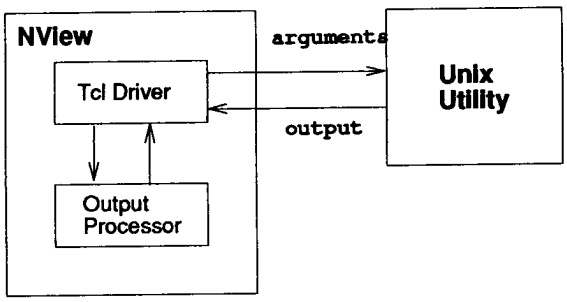


Figure 3: Interaction With Other Utilities

a procedure which is able to interpret output from the utility and use it to perform some meaningful function.

When the utility is invoked, the driver creates a new socket to be used to collect output from the utility, associates the socket with a specified output processor, and finally forks the utility as a child. The output from the utility is redirected to the aforementioned socket and led back to the driver (see Figure 3). The driver collects output until the utility terminates, at which time the associated socket is closed and the collected output is passed to the output processor. The output processor then has complete freedom to access the display, topology database, and other utilities in order to process output in a useful manner.

The advantage of this method over an agent acting as an intermediary between the driver and the utility, is that input parameters are decoupled from the output format. Using a single module to provide bi-directional communication would require a separate agent to deal with each utility and type of output. On the other hand, an output processor only deals with output, so general-purpose processors can be written suitable for use with many different utilities. We will see one example of this in Section 3.1.3 below.

Conversely, a single utility, such as **netstat**, may be able to generate many different output tables. Typically, a separate output processor would be written to deal with each type of output. When a new output format is added to the utility, one need not change any existing output processors, but may simply create another output processor to deal with the new type of table.

Since Tcl handles variable expansion in arguments, there is no need for input translation as long as everything can be specified on the command line. NView does, however, allow shell features such as pipes, wildcards, and input redirection.

For example, using this level of integration, one could call **traceroute** to quickly generate output showing IP addresses only, and then fill in the hostnames from the network database as they are displayed. This way, **traceroute** need not spend time looking up hostnames, which can be slow. At the same time, the output processor for **traceroute** could also update the network database with the new information about physical links.

The integration work required here consists mainly of writing the output processor to process the output of the utility. Once this is written, the programmer assigns

the processor a built-in name, and NView is recompiled. Macros can then be added to start up the processor with specified commands. For example, if `traceroute`'s output processor is called "parse_route", the user may define a macro to "`nv_start parse_route traceroute -n -g $fromip $toip`" to initiate a traceroute to the destination which crosses the currently selected application-level link and have the output processed by the parse_route routine.

#### 3.1.3 Level 3: Integration With No Output Interpretation

In the third and loosest level of integration, the function of the utility may be completely independent of NView. The output of the utility has no impact on the NView database or other NView modules. In this case, a default output processor is used which simply displays the output of the utility in a separate window. For example, the user may wish to define a macro to invoke `finger` to simply list users on the currently selected host, such as "`nv_exec finger @$hostname`".

No changes to NView are needed to integrate a utility at this level. Integration may be done by defining a macro in a startup file, or even during run-time, or by executing the utility as a full command line inside NView.

## 4 Network Monitoring and Visualization Features

There are many important considerations in effectively visualizing and managing networks, including how to break down the graph into logical sections, how to display information, how to cover both applications as well as the underlying network topology, and how to ensure information is recent. Each of these will be covered in turn.

### 4.1 Network Decomposition and Representation

Various techniques exist for laying out graphs in an understandable manner, such as the Spring Embedder algorithm described by Fruchterman and Reingold[9]. Typical goals are to distribute vertices evenly around the graph, minimize edge crossings, and make edge lengths uniform. In achieving these goals, there is an obvious trade-off between aesthetics and speed.

When the number of nodes in the graph becomes large, it becomes computationally intractable to achieve an attractive graph in an interactive application. Indeed, it may even be impossible to draw every node in any sort of understandable fashion. Also, by the time we could collect the necessary information to cover all the nodes, the information obtained may be far out of date.

Thus, it becomes necessary to reduce the number of nodes to be displayed at a time. NView allows the user to select from several methods of decomposing the network into a collection of components. Other methods can be easily added as the need arises. For instance, NView allows the user to logically partition the network for display based on the network's topology. We define a *component* to be a set of strongly connected nodes with no *cut edges*. A cut edge is any edge which, if removed, partitions the graph. We define a *subtree* to be a set of connected nodes with no cycles. Typically, networks are composed of both components and subtrees.

The sets of edges in components and subtrees are then always disjoint, but a node may be part of both a component as well as a subtree. Given a network with $E$ edges, we can easily break down the network into its components and subtrees[3] in $O(E)$ time. This is done by using a backtracking algorithm to mark edges which are found to be part of cycles. This decomposition makes the topology of the graph more explicit.

Another method for logically breaking down a network into understandable components is division by domain. For instance, administrators over a network could display and monitor all the nodes on their particular network. However, this approach is not sufficient for monitoring applications which span multiple domains. One could limit administrative scope to a set of specific subnets or even selected countries, in some cases. In the general case, however, one may have a set of distributed applications, such as the World Wide Web, which spans many administrative boundaries. Therefore, it is desirable to allow the user to choose the most appropriate method of decomposing a complicated graph.

### 4.2 Logical Overlays

It is important in managing distributed applications to understand the precise topology of application-layer connections and their relationships with the underlying physical links. Too many high bandwidth streams across a single physical link can cause a bottleneck. It is thus important to be able to visualize the relationship between the two layers of connections.

NView addresses this problem by providing the ability to overlay a map of application-layer nodes and links over a map of physical links and intermediate nodes which make up the physical layer. In this manner, both bottlenecks as well as unused links become visible. Since NView provides facilities for displaying the topological effects of adding, deleting, or modifying the characteristics of individual nodes and links, the administrator can design a better topology before making any actual changes to reconfigure networks.

Another benefit of logical overlays is the ability to see the effects of physical-layer problems on the application-layer. This makes it easy to identify critical points of failure, as well as track down the cause of problems when traffic between application processes is disrupted.

[3] For example, see figures 4 and 5, which will be discussed later in section 5.3.

### 4.3 Display Criteria

Once we have determined the extent of the network to display, we need to turn to the question of *how* best to display it. The particular information in which the user is interested may have to be obtained via SNMP or some other application-specific protocol. In a general tool, the user must be able to request arbitrary information.

Administrators often want answers to a variety of questions. For example, one may wish to identify nodes which support SNMP and have no hostname registered with DNS. NView displays the graph in such a way that the answers become obvious from looking at the display, and the relationships between the nodes or edges are readily apparent.

Once the information is obtained, the nodes and edges are made to represent the information by symbol, size, or color. Using multiple features, nodes and edges can visually reflect multiple pieces of information at a time.

### 4.4 Timeliness

An important factor in the success of a monitoring and debugging tool is the timeliness of information. Optimizing for speed is thus important. For example, updates should be asynchronous, so that processing can continue in the background. For effective network management, it is also useful to allow events to set off alarms or trigger actions such as alerting an operator, or taking corrective measures where possible.

NView allows commands to be set to be executed periodically. An example is to update all nodes in the currently displayed component at regular intervals. An event list is kept containing the command to execute, period interval, and next time of execution. As each event's time arrives, the command is executed and then rescheduled to occur after the specified interval.

Alarms can be linked to specific pieces of information about a node or link. An alarm consists of a trigger condition, rearm condition, initial status, and a `Tcl` command. An alarm's status can either be active or inactive. When active, the command will be executed whenever the trigger condition becomes true, and the status will change to inactive. The status will remain inactive until the rearm condition becomes true, whereupon the alarm will become active again and ready to be triggered.

## 5 Case Study: The MBone

The MBone (Multicast Backbone)[10, 11] is a virtual network running over the Internet, and is used to broadcast multimedia traffic across the world. MBone nodes currently use IP tunnels to relay information to sites, which then use multicasting to deliver packets to users.

Since March 1992, the MBone has grown from an initial 40 subnets to approximately 1000 subnets. Given the current explosion of MBone use and the need for management tools, the MBone provides an ideal example of an application layer to which NView can be put to immediate use.

In the MBone, a multicast routing daemon (**`mrouted`**) runs on each host which is considered to be part of the MBone itself. Each **`mrouted`** communicates with other **`mrouted`**'s using stream connections, resulting in an application-layer of connections sitting on top of a physical topology. Other hosts can communicate over the MBone by communicating via multicast with an **`mrouted`** on the same subnet. However, these hosts can be thought of as clients, and will not be considered part of the management model for purposes of this discussion.

Monitoring and managing the MBone presents a number of problems. The vast majority of nodes do not recognize SNMP so a series of piecemeal utilities have sprung up to deal with different problems. No automated scheme is available for finding the best topological change to accommodate new nodes. Until now, this task was done manually by administrators looking at hand-drawn network maps. It is also difficult to track down the source of serious problems, such as when half the network suddenly loses a signal.

### 5.1 Example Scenario: PIM routers

In late March 1994, it was discovered that the routing tables in the **`mrouted`**'s on the MBone were overflowing. While only about 1000 subnets were actually on the MBone, the **`mrouted`**'s were found to be holding up to 8000 subnets.

Soon, the problem was traced to Cisco's beta-test PIM (Protocol Independent Multicast) routers, a new product combining unicast and multicast routers. These PIM routers were mistakenly reporting all subnets as supporting multicast, rather than only those on the MBone.

On April 7, Dino Farinacci at Cisco requested that all beta-test sites temporarily disconnect from the MBone until the problem was fixed. When this was done, it was found that the problems continued. Apparently, there was at least one more PIM router unaccounted for, and it became critical to track it down.

At the time, the solution administrators chose was to shut down the MBone for a day and partition it to see which areas became stable and which continued to suffer problems. For approximately the next 24 hours, this search continued and the problem was narrowed down to somewhere in Europe.

To complete the description of how this problem was ultimately solved, we will first need to review the MBone management tools that were in existence. We will resume this account in Section 5.4.

### 5.2 MBone Tools

A number of relevant tools exist for the MBone in particular. Van Jacobson's `mrinfo` can retrieve information such as link types, distance metrics, TTL thresholds, and current status from a specific **`mrouted`** node. However, the utility is slow since it must look up the host name for each multicast router mentioned.

Somewhat more efficient is **map-mbone**, Van Jacobson's improvement of Pavel Curtis' original mapping program, which generates a file describing nodes and links in the MBone.

Atanu Ghosh's **mwatch** tools maintain a centralized server containing the current topology, which is updated about every two minutes. The client program which queries the server is a text-only utility, and is useful for keeping network traffic down. Unfortunately, the times when the server is likely to be most useful are when the network is partitioned. At such times, the server may be unreachable or have incomplete information. A centralized server model is clearly inadequate.

An off-line debugging tool, **mrdebug**[12], by Deborah Agarwal, uses an **map-mbone**-style file to allow the user to model the effects of proposed topological changes on source broadcast trees and source-to-destination paths. It has proven useful in solving many MBone problems, but has the limitations of being neither visual nor on-line.

### 5.3 MView: An Instantiation of NView

MView is an instantiation of NView for the MBone. It automatically discovers all topology information without relying on setup files or manual user configuration and saves information between sessions. It is completely autonomous and can be run from anywhere on the Internet, even from hosts not on the MBone. However, since MView uses **map-mbone** to get information on a node, the kernel must support multicast packets in order to use MView as an on-line monitoring tool.

MView can break down the graph into components and subtrees and identify the largest component as the default *backbone* from which other subtrees and components extend. The user may temporarily change link statistics and add new nodes and links in order to see the effects on the topology of the MBone and the paths which traffic will take.

Figure 4 shows an example of MView at startup time. The main window displays the current component, while the window on the right displays information specific to the highlighted node, including distance from a specific source (if any), **mrouted** version number, SNMP version number, status, hostname, IP address, and the statistics and status of all links defined. In this figure, nodes are being shown by domain.

In the node window on the right, we see information about the current status of the display and the node mbone.merit.edu. In particular, we see that its current display coordinates are (186,293), its node number is 1, the distance from the specified source is 32 (no source was specified), the current component number is 1, and there are 32 nodes displayed. The next line gives us the current node, component, and subtree numbers for the selected node, the mrouted version number (2.2 for mbone.merit.edu), and various status information. The current node is responding to pings, has multicast ability, is running an **mrouted**, and does not recognize SNMP.

Link statistics include metric, threshold, status and the hostname of the other end. The user can display either incoming links or outgoing links. Other nodes can be displayed by clicking on the node in the main window, or by clicking on a link to follow in the window on the right. Various other mechanisms exist to locate a specific node according to some criteria.

For an example of link information, refer to the line entry for taipei.eecs.umich.edu in the figure. Here we see that this link has metric 1, threshhold 1, is an IP tunnel, is currently down, and the link is contained in subtree 2 so is not currently on the display, which is showing component 1.

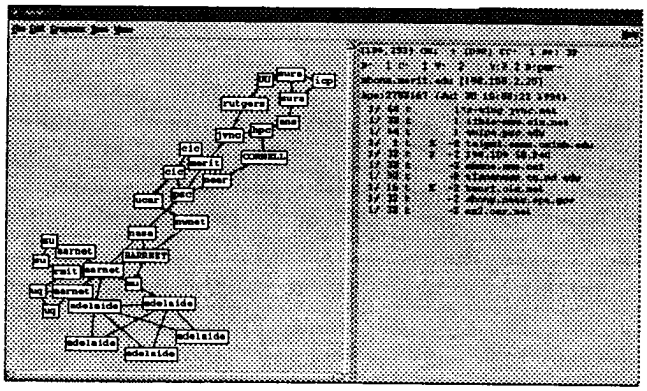

Figure 4: Sample Component

Figure 5 shows a sample subtree. Connecting links are being displayed whenever the link between two nodes is operational. "Floating" nodes are those which have links to the rest of the MBone configured, but whose tunnels are all currently down. In this figure, nodes are being displayed by **mrouted** major version number. Note that PIM routers will have no **mrouted** version number.

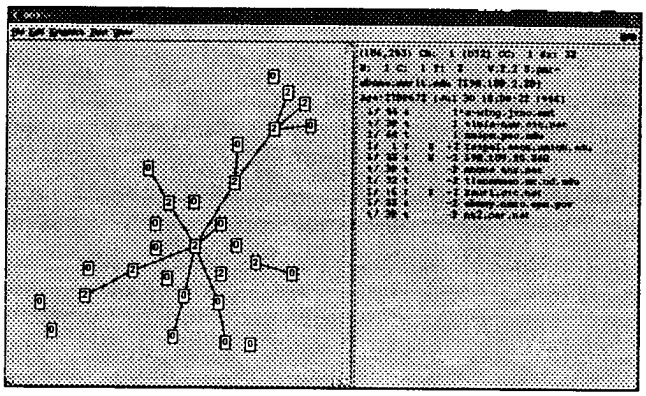

Figure 5: Sample Subtree

MView allows displayed nodes to be decorated with attributes such as version, SNMP version, name, or domain. All links can be displayed, or one may view packet distribution by limiting the displayed links to those in a source tree, or those along a source-destination path. Displayed links can also be limited to those below a specified TTL threshold.

Nodes may be highlighted by such criteria as SNMP avail-

ability, whether **mrouted**'s are responding, age, whether they are receiving, and the existence of links to another component. Links can be highlighted according to various criteria, including whether they connect to an unresponsive node, and whether they are in a selected source tree or source-destination path. For convenience, all temporary changes made as part of "what if" experiments can also be highlighted.

The functionality of a number of utilities are already incorporated into MView, including **ping**, **traceroute**, **snmpget**, **nslookup**, **mrinfo**, and **mrdebug**. Other cabilities are relatively easy to add.

### 5.4 PIM Scenario Solutions

We now continue with the MBone catastrophe begun in section 5.1 above. The MBone had been non-operational for almost a day and work was progressing slowly at first. Finally, Van Jacobson pointed out that PIM routers would not respond to **mrinfo** probes (since they do not run **mrouted**), but would be reported as up by their neighbors.

Atanu Ghosh and Piete Brooks then used the **mwatch** server in the U.K. to help identify a renegade PIM router in Sweden. Administrators there were then contacted and the router was disabled. The problems then resolved, the "all clear" was given and the MBone resumed operation as the various pieces were manually reconnected.

Were this same situation to occur again today, we expect that any administrator could use MView to identify the problem in minutes, if not seconds. For instance, MView allows the user to locate and highlight nodes which are reported by neighbors to be up, but which do not respond to queries. This would help identify PIM routers.

Similarly, arbitrary SNMP queries can be done to retrieve information from nodes which recognize SNMP. Other types of queries can easily be added as needs arise.

## 6 Conclusions

We have described NView, a visual framework for network tool integration. We have also implemented MView, a prototype of NView for use with the MBone. MView provides a visual framework for managing and monitoring the MBone and incorporates the functionality of many other useful tools. It addresses both the problem of managing the connections of the "application" layer, as well as displaying the links of the "network" layer. Together, this provides a convenient tool for both network debugging and design.

As SNMP becomes more prevalent, the importance of SNMP compatibility will increase. A proposal is now underway to support multicast information in SNMP for physical routers that can do multicasting[13], such as the PIM routers. It is expected that our tool will become even more powerful when that happens.

NView is also useful in a far broader context than our example, the MBone. This power derives from the framework it provides for incorporating new partitioning algorithms, its methods for acquiring information, and the convenience of invoking its commands.

## References


[1] William Stallings. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network-Management Standards.* Addison Wesley, 1993.

[2] Marshall T. Rose and K. McCloghrie. Concise MIB definitions, March 1991. RFC 1212.

[3] Keith McCloghrie and M. Rose. Management information base for network management of TCP/IP-based Internets: MIB-II, March 1991. RFC 1213.

[4] Marshall T. Rose. SNMP MUX protocol and MIB, May 1991. RFC 1227.

[5] B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, and G. Waters. Simple network management protocol distributed protocol interface version 2.0, March 1994. RFC 1592.

[6] Douglas E. Comer. *Internetworking With TCP/IP Volume I: Principles, Protocols, and Architecture.* Prentice Hall, 2nd edition, 1991.

[7] J. H. Chou, C. R. Buckman, T. Hemp, A. Himwich, and F. Niemi. AIX NetView/6000. *IBM Systems Journal*, 31(2):270–285, 1992.

[8] Robert S. Cahn, Pao-Chi Chang, Parviz Kermani, and Aaron Kershenbaum. INTREPID: An integrated network tool for routing, evaluation of performance, and interactive design. *IEEE Communications*, July 1991.

[9] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice & Experience*, 21:1129–1164, Nov 1991.

[10] Steve Casner. Frequently asked questions (FAQ) on the multicast backbone (MBONE), May 1993. Available as ftp://venera.isi.edu/mbone/faq.txt.

[11] Michael R. Macedonia and Donald P. Brutzman. Mbone provides audio and video across the Internet. *IEEE Computer*, April 1994. Available as ftp://taurus.cs.nps.navy.mil/pub/mbmg/mbone.ps.

[12] D. A. Agarwal and Sally Floyd. A tool for debugging Internet multicast routing. In *Proceedings of the 22nd ACM Computer Science Conference*, pages 22–29, Phoenix, AZ, March 1994.

[13] T. Pusateri. Managed objects for the IP multicast forwarding table. Internet draft, March 1994. Available as ftp://venera.isi.edu/internet-drafts/draft-pusateri-ipmulti-mib-00.txt.