Homebrew Network Monitoring: A Prelude to Network Management

Mike Schulze, George Benko and Craig Farrell
Department of Computer Science
Curtin University of Technology
Perth, Western Australia
mike@cs.curtin.edu.au, rete@cs.curtin.edu.au, craig@cs.curtin.edu.au

March 12, 1993

Abstract

A wide variety of public domain and commercial tools exist to help network administrators manage networks. Few of these tools achieve a satisfactory price/performance ratio for organisations with small budgets. To date, we have implemented a number of tools which allow us to examine and visualise network communications with a simple, intuitive, X based graphical user interface. These tools acquire knowledge on network communications via passive network monitoring, with a minimal amount of user intervention. This allows users who do not have an in-depth understanding of network architectures to gain an immediate, intuitive understanding of their network and its performance.

1 Introduction

With the sudden growth of UNIX™ based networks, a need for effective network diagnostic tools has arisen. Indeed, many tools have been implemented to help solve this problem, but are beyond the financial reach of network managers within small organisations.

Management of a small to medium sized network need not be an expensive, complex undertaking. Some of the functions provided by specialized network management packages can be implemented by a sub-set of publicly available tools. Of these tools, some are simple load monitors, others, such as *etherfind*[33] and *Tcpdump*[16] provide in-depth textual descriptions of local network traffic. These utilities may be sufficient for an experienced user, but can become cumbersome and/or uninformative in the hands of the uninitiated.

In this project, we attempt to extend the goals of these utilities by visualising network data. This has been achieved by applying a graphical model to a collection of continuously updating network statistics. These statistics are gathered by promiscuously monitoring the local network.

In the early stages of development, a number of tools emerged which collectively form the "Netman" project. The primary goal of the project is to provide an intuitive representation of network activity using graphical techniques. Several of the Netman tools have been inspired by non-text based network monitors like *EtherView*[14], *NetVisualyzer*[29], and *traffic*[36]. It is envisaged that Netman tools will **supplement** information provided by other monitoring tools rather than replace them.

The development of this project was entirely under UNIX™: we have not considered other operating systems[1]. At present we have implemented our work on Sun SPARCstation™ and DECstation™ 5000 series architectures. These platforms will be referred to as monitoring stations in the remainder of this paper. We chose to implement our work on these workstations for reasons similar to those outlined in [24].

All of our efforts are concentrated on monitoring Ethernet™[22] networks. A typical university ethernet, such as ours, provides us with a good environment in which to develop our project as it exhibits a wide variety of protocols.

There are two methods of passive network monitoring: real time analysis and retrospective analysis[2]. Our initial work concentrated on real-time traffic analysis, which forms the bulk of this paper. To a lesser extent, we explore an implementation of retrospective analysis in the form of a protocol analyser.

2 Statistics via Passive Monitoring

Passive network monitoring can provide a good insight to common network configuration problems.

1. This is not to say that network traffic generated by hosts not running a UNIX derivative will be ignored.

Examining the "wire" can often reveal problems such as protocol mismatches, incorrect routes, broadcast storms etc. We chose the passive approach because we were interested in monitoring normal network operations without interference from either a custom or established monitoring/management[2] protocol. That is, the act of observing a network should not directly interfere with or add to network activity.

Having access to local network data allows us the freedom to provide a multitude of statistical information. In this way, we are limited only by the amount of network data we can process "on the fly". A combination of specialised data structures and a reasonably reliable network tap forms the basis of the monitoring engine.

2.1 Packet Capture Mechanisms

User level access to physical network traffic requires an efficient mechanism which is able to interact with the monitoring station's network interface device drivers. Some modern implementations of UNIX have such a mechanism, e.g. Sun's *Network Interface Tap* (NIT)[34] in SunOS™, *Snoop*[30] in IRIX™, and the Ultrix™ *Packetfilter*[9]. All of these implementations are derivatives of the original "packet filter" developed at Carnegie-Mellon University in 1980[25]. At present we use the Ultrix packetfilter and NIT under SunOS. Although there has been a recent paper[19] outlining the ineffiencies of these mechanisms, they are standard system resources and provide sufficient performance and functionality for our work.

Data is captured by placing the network tap[19] into promiscuous mode and collecting statistics from the incoming packet headers. Information from each buffered queue, or "chunk" of packets, is decoded sequentially and distilled into the appropriate statistical data structure. No attempt has been made to make the packet decoding routines protocol independent; all protocol specific information is hard coded.

2.2 Determining Logical Network Connections

Finding connections between hosts forms the basis of our work. The method is simple: examine the source and destination fields of each passing packet. Once a source / destination address pair is obtained, it is stored in a continuously updated list of connected nodes. Each node which is identified as a source of network traffic, will have an associated list of connections to destination nodes. Both lists are updated as often as possible, in our case after each chunk of packets is read.

A node is never deleted from the list. Therefore the list is limited only by the number of hosts on the network and the system resources of the monitoring station. An idle[3] host is not removed from the list, instead it is flagged as idle. This way, if the host were to re-transmit, the idle flag could be toggled and the previously determined host statistics would still be available.

2.3 Data Flow Calculations

In order to extract useful information from network traffic, we have outlined a set of statistical categories which we consider important:

- per host source traffic flow
- per host protocol summaries
- per link traffic flow
- per link protocol summaries
- overall bandwidth consumption.

These calculations must be made each time a chunk is processed. This means that we must maintain a list of running totals each time a packet is decoded and then perform the calculations.

After each chunk is processed, a *chunktime*[9, 34] or time frame is established by finding the difference between the timestamps associated with the first and last packets in the chunk[4]. This value serves as a point of reference for all flow statistics generated from the running totals.

Per host source traffic totals are tallied by examining the source and length fields of each packet, then incrementing byte and packet totals for the corresponding node. These totals are then divided by the chunk time to yield a result which is stored in a sliding buffer of finite length. From this brief history of traffic statistics we can obtain a "sliding average" which is used to represent source traffic for that node. Link statistics are handled in a similar way, as running totals for link flows are also maintained.

Protocol summaries are more difficult: depending on the protocols considered most interesting to the user, each protocol layer must be decoded to the required level. For example, providing the frame type[8] of each Ethernet packet is not very useful if the network is predominantly IP[28]. For this reason, we have introduced extra decoding routines that will provide summaries for UDP and TCP port connections. Although this involves more work for the monitoring station, we believe that the usefulness of this

2. We are not attempting to criticise SNMP[4] in any way. The goal of the Netman project is to provide statistics on normal network operations as opposed to providing a management infrastructure. Although we realise SNMP MIBII[20] partially provides the monitoring functionality, it is lacking in per protocol based statistics.

3. The term "idle" is used loosely; the user must decide the parameters which class the host as idle.

4. This value is slightly inaccurate because we ignore the time gap between consecutive packets separated by two chunks. Although we realise that this value is not correct[24], it is sufficient for our purposes.

information outweighs the extra load.

Producing these summaries requires a moderately complex tree structure which is associated with each link[5]. The root of the tree is a pointer to a list of Ethernet protocols being used on a link with its associated byte/packet totals. Each frame type may have an associated sub-protocol, so in the case of IP, sub-protocols may include TCP, UDP etc. Following in this tradition, each sub-protocol may have a sub-sub-protocol, which we have termed the *port type*, solely for the TCP/IP suite. These statistics are running totals which are never converted to flow rates, but simply updated after each packet.

2.4 Limitations of Passive Monitoring

Restrictions to passive monitoring means the Netman tools can only access data on the local segment and other networks remain "invisible", i.e. cannot be monitored. Sun has attempted to solve this problem by writing a simple rpc service, *etherd*[32], which allows a monitoring station to connect to a host on another network and retrieve traffic statistics from the remote network. Unfortunately, our requirements were more complex than that of calculating a bandwidth usage statistic every second. We considered writing our own customised rpc service. However, this would have contradicted our requirement for passive monitoring, as such a service would inevitably generate a significant amount of additional traffic. Silicon Graphics takes a similar approach with *snoop*.

One of the limitations of passive monitoring is not being able to request information from the network. For example, SNMP will allow a management station to query a device (which is running an SNMP agent) to retrieve the requested MIB variables. All that can be obtained from a monitor are cumulative statistics concerning the active devices on the local network. This means that simple questions like, *is my host up*? cannot be answered if that host is idle[6].

Another inherent limitation on monitoring is that we are unable to detect physical level devices such as repeaters. That is, a fault occurring on a repeater segment, will only be detected via inability to reach hosts on that segment. Of course a traditional method can be used, such as sending an ICMP echo and waiting for a reply.

2.5 Effective Monitoring Strategies

The use of bridges can restrict the amount data which a monitoring station can access. A local Ethernet may be viewed by some as a single length of cable rather than a conglomeration of segments and bridges, but the use of a bridge[7] to fragment a network may serve as a "firewall" to the monitoring station (see Figure 1).

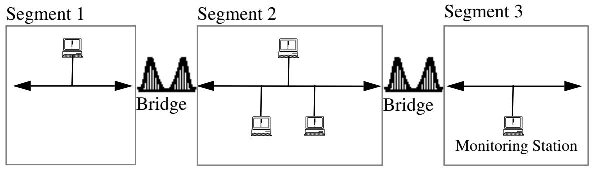

Figure 1. Using bridges to segment an Ethernet.

In this case the monitoring station will only see the traffic on or destined for segment 3. That is, even though the traffic on segments 1 and 2 may be considered local, the monitoring station can only see what is on the immediate segment. Traffic from segments 1 and 2 will appear to come from the ethernet interface of the bridge connected to segment 3.

Depending on the network activity that is considered most important, the monitoring station must be placed on a segment which is most likely to have the highest proportion of interesting traffic. For example, if remote IP traffic was of major concern, the monitoring station would be best placed on a centralised backbone which has a high exposure to foreign traffic.

2.6 Assumptions made on Physical Transmissions

Accessing data at the link level remains a problem, even if the workstation is capable of processing network data as fast as it arrives. The information being transmitted on the physical Ethernet will not always appear at the link level regardless of machine processing speed. The physical level hides collisions, checksums (valid or otherwise), corrupted frames etc. from the protocol level accessed.

Calculated load average, which represents the actual level of network traffic, must be as close as possible to the correct value. This means that certain assumptions about the data received are required. For example, a received packet is assumed to have had an associated preamble, frame check sequence, and an inter-frame gap[1]. This assumption compensates for the fact that these pieces of "lost" information are left at the physical level at the time of transmission (see Figure 2). Some network monitors fail to take this into account, creating a slightly inaccurate calculation of bandwidth usage, e.g., *xtr*[8][27].

---

5. It is not the intention of this paper to provide a detailed account of the internal data structures we have implemented, so the discussion will be limited to a general overview of the statistical structures. Although it is useful to note each node has an associated list of links, and summaries for each node can be derived from this information.
6. Although *rwhod* will make a host "visible" if broadcasts are being monitored.

7. Most modern bridges or level-2 routers[38] will only **route** packets if the destination Ethernet address is located on the other side of the bridge relative to the source device.
8. Unfortunately an assumption of 10 * 1024 * 1024 bits/sec was also made, which is not the correct speed of 10^7 bits/sec.

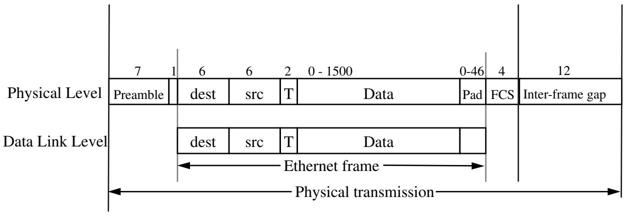

Figure 2. Frame lengths at different levels.

There is no facility within the interface "tap" to detect Ethernet collisions, therefore there is no way of determining the amount of information transmitted during a collision sequence[9]. This is yet another contributor to the overall problem of not being able to represent network traffic accurately.

Truncated or "runt" packets, that is packets with a length less than 60 bytes, should be treated as normal packets. This depends on whether the interface tap will pass the packets as valid transmissions. Since we have never encountered such a packet, we cannot confirm whether they are treated as a normal transmission. Misaligned packets have not been considered.

2.7 Packet Filtering

Packet filtering is the ability to focus on a specific protocol or a set of protocols, whilst the physical network interface is promiscuously capturing network data. Most network taps are able to filter packets at data link level and although Ultrix Packetfilter and NIT have this facility, we do not make use of it in our implementation. However, in order to avoid potential future problems when dealing with differences between filtering schemes across would-be platforms, and to provide access to all network traffic (used for calculating bandwidth statistics), we have implemented our own set of filters.

3 Visualising Network Data

One of the goals of this project was to provide tools with graphical representations of network traffic that are easy to understand, yet complex enough to retain all of the relevant information. Although it is hard to say if any single graphical representation could be useful in all situations [24], we believe a depiction exists for network communications which involve source/destination pairs. The remainder of this section outlines a model which is general enough to display overall network communications in a given network.

9. Although there is a way of determining the number of collisions on the interface since the machine was booted, this information is not of much use if a time-stamped collision indicator cannot be obtained.

3.1 Modelling Network Traffic

Our approach to traffic visualisation is to create a graph of network activity. To provide an accurate model of the network, the nodes, depicted as circles, represent devices, and edges, shown as line segments, represent logical connections between network devices. This graph is *connected*[37], that is any pair of nodes are directly reachable from one another.

Since typical networks have many devices sporadically but frequently communicating with each other, a representation of the network quickly evolves into a sprawl of lines and circles unless there is some form of logical node ordering and labelling. We chose to arrange the graph in a circular manner, with all nodes equidistant (this idea is similar to the approach taken by *NetVisualyzer* and *Etherview*). Each node has a unique label which identifies it from other nodes. The positioning of the label follows a perpendicular to the tangent of the network "circle" at the node position (see Figure 3).

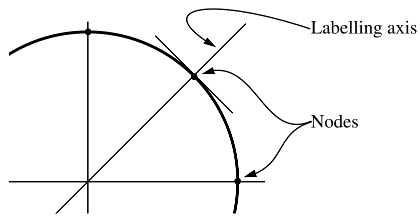

Figure 3. Node labelling and distribution.

A problem arises when we consider that each edge of the graph is undirected. That is, one host could send data destined for another, and the destination does not respond. Our model will create a connection between the two nodes but no indication of direction will result. Initially, we considered this a problem until we decided to identify the source and volume of traffic for each node.

Each node represents a network device which generates intermittent transmissions. The area of the node is a function of the source traffic generated by that device. That is, the source traffic for each device is directly proportional to the area of the circle, so devices generating a higher proportion of traffic will be more prominent.

Since each device may have multiple connections, line thickness may be used to represent the amount of traffic on a link between two hosts. Other packages, such as *EtherView*, display traffic statistics between hosts by a colour coding scheme. We have found this difficult to understand when the display update time was less than a second, and more than a few colours were used. *NetVisualyzer* uses colour intensity to represent traffic flow on a link, that is, a single colour is used and only the intensity of the link is varied. We believe our approach is more intuitive; if the link gets wider, more data is being transmitted.

The final aspect of our model involves the identification of *dominant* protocols on each link. A protocol is considered dominant if it has the greatest byte aggregate from all protocol transmissions for any given link. A pre-determined colour table assigns each protocol with a configurable colour code. Once the dominant protocol has been determined, the colour table is consulted and link colour applied.

So far we have assumed a network protocol that is bound to local communications only. If communications between hosts on differing networks are accessible[10], an augmentation of our model can be used to group hosts into a logical ordering of networks. This makes it possible to represent communications within network layer protocols, such as IP or DECnet.

We feel that the overall model just presented, is sufficient to provide a good picture of arbitrary network communications. We also feel that the model should not be expanded past its present form, for fear of introducing redundant complexities. For example, if arrows were used to imply direction of data flow on a given link, in most cases this information would be redundant as the majority of communications will involve a valid source / destination pair with bi-directional flow.

The question of whether or not certain statistics are relevant on a crowded display will always cast doubt on our claim that the model is general enough to show all network connections. In practise, we have taken measures to avoid screen overcrowding by implementing host/network time-out and compression mechanisms.

3.2 Animating the Model

Providing instantaneous feedback of network activities is an essential part of real time fault diagnosis. The original implementation of our model was conceived on a Silicon Graphics 4D/20 workstation using the native *gl*[31] library. This platform provided the necessary graphical resources and the network tap we required to produce a fast working implementation.

The *gl* library made it possible to develop a set of display routines which simply redraw the network, as often as possible. That is to, gather network statistics and redraw the network *ad infinitum*. This version was only capable of monitoring connections between network devices.

Once we were satisfied that the code had matured enough to warrant porting, it was decided to implement the code using the X11 Windowing system[23] to ensure graphical portability across platforms. The prospect of converting our carefully crafted display routines to *Xlib*[10] calls prompted us to search for an alternative method. After careful consideration we decided that we would use the *vogl*[13] library to integrate X with the *gl* calls we had grown fond of. Fortunately we had only made use of a few *gl* calls, all of which were supported by *vogl*.

3.3 Display Implementation

Bringing our ideas to life proved to be a balancing act between monitoring station resources and portability considerations. Our first implementation allowed much freedom when it came to animation, but lacked portability. Since we wanted to create a tool which contained our animated display as well as menus, push-buttons, graphs etc., we decided to explore the *X Toolkit*[21] to help solve our problem. Fortunately, we were able to implement a working version using *Xt*, *vogl* and the *hershey*[12] font libraries[11]. Initially we chose the *Athena* Widget set from MIT[26] as it is publicly available and provided the necessary Widgets we required. Currently, the Xaw3d[17] Widget set is used, purely for aesthetic reasons.

The present implementations of our work provide us with a set of robust, informative monitoring tools. Surprisingly, the display is dynamic enough to show network traffic in real time, without any significant lag between burst and display. Unfortunately this is not the case when the X display and the monitoring station are several networks apart.

4 Implementations

A need for informative, easy to use network monitoring tools within the international UNIX community was becoming more apparent. This lead us to experiment with public domain and system resident tools which have helped us to achieve what we have. Most of the tools that we found are text based, e.g. *NNStat*[3], *Tcpdump*, *etherfind*, *nfswatch*[5] etc. and provided real-time statistics on network traffic. Although these tools have proved invaluable, we felt a graphical approach had far more to offer in terms of providing a user with a picture of all network traffic.

4.1 Monitoring All Ethernet Traffic: *etherman*

Our first implementation was directed at monitoring all ethernet host connections in real-time. Later, we added support for displaying the amount of source traffic from each node, and the amount of traffic on each host pair link. After testing *etherman* on different networks, it became obvious that we needed to be able to concentrate on "important" network activity by adding time-out and scaling functions. For example, a quiet network will appear

10. For example, Monitoring IP communications can often reveal source / destination pairs that do not reside on the local network.

11. The *Hershey* libraries were necessary because *vogl* does not support any form of text display. This also provided the font rotation which was available under *gl*.

static or uneventful, but it is possible to scale nodes and links to allow a better picture of relative network communications.

Network statistics are produced in both textual and graphical form. Dynamic statistics such as host and link traffic flow are displayed via our graphical model of network communications in real-time (see Figure 4). If a particular network phenomenon results during the course of execution, a postscript™ snapshot of the network is possible. Bandwidth consumption statistics are sampled once every second and displayed via a scrolling stripchart widget.

A textual dump of protocol summaries collected for each host and link is available. The information given is for the run-time of the program. These summaries are currently decoded for all Ethernet frame id's and all IP protocols. The output format for each host pair indicates the amount of data exchanged in bytes and packets, and the protocol in use.

We have used *etherman* to detect a variety of network problems and shortcomings. Because of its graphical nature, problems such as excessive bandwidth consumption and broadcast storms become easy to identify. It is also very useful for finding unexpected transmissions or unknown devices.

4.2 Focusing on IP Connectivity: *interman*

Using the augmented display model we can monitor a network level protocol, in this case IP, to display connectivity and dominant sub-protocols. In this way, networks are ordered in a circular manner much the same way as Ethernet devices in *etherman*. Unfortunately, this model will not display routing information as IP routing is transparent at this level.

As with *etherman*, we monitor and display traffic in real-time with update times varying because of traffic flow and monitoring station capabilities. As can be seen from Figure 5, different networks are shown as circles with hosts

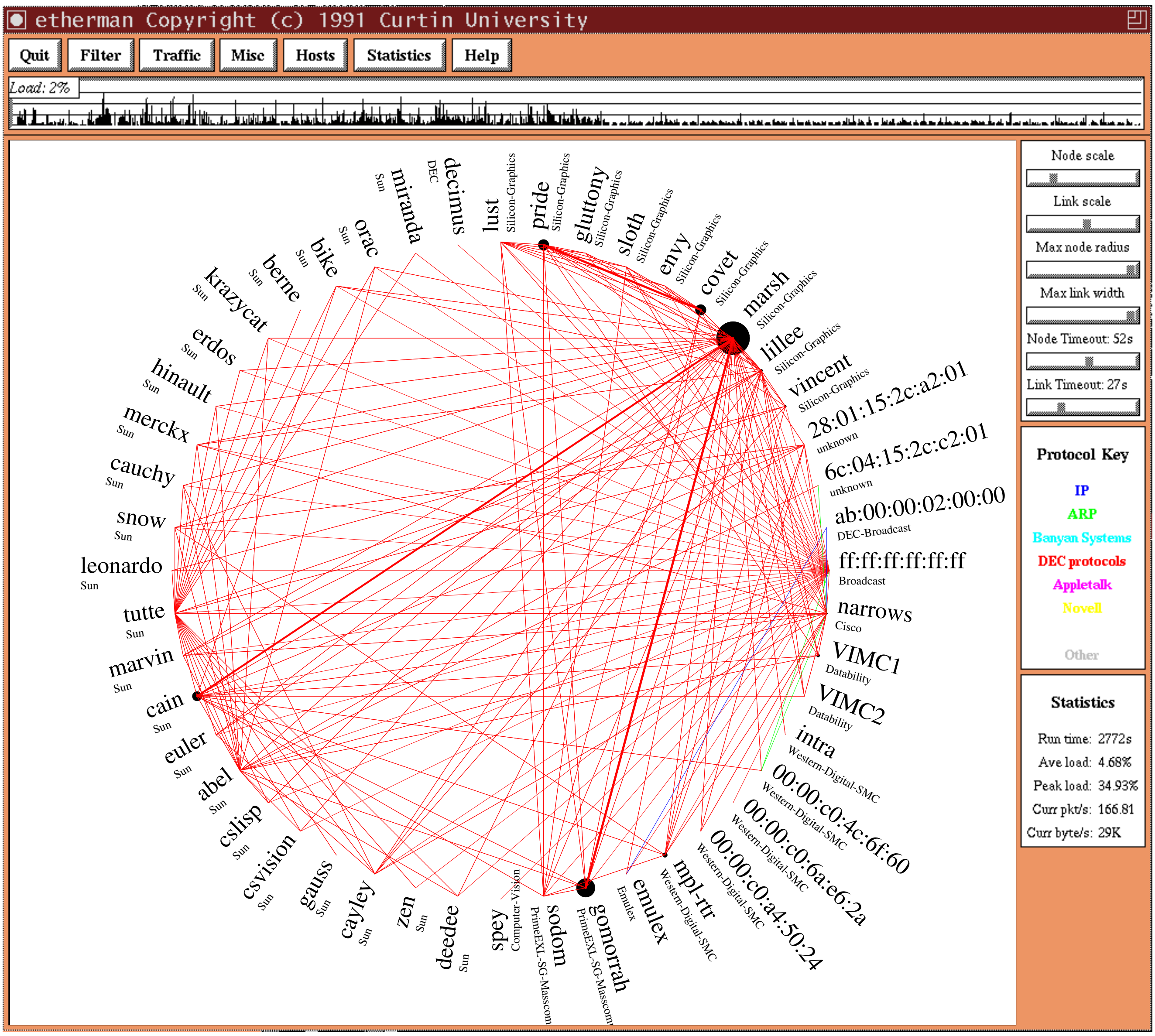

Figure 4. *etherman* monitoring all local traffic

local to each network listed around their respective circumferences. Hosts are connected between two networks via a single link; the colour of the link denoting the dominant protocol. Hosts and networks appear on the diagram as communications are monitored, and disappear when a host or network has been idle for a configurable period of time.

Figure 6 shows a sample summary of selected transmissions for host *cujo* on a per host, protocol breakdown. Since these summaries are derived from link statistics, we provide all communications involving the said host. This means that some information will become redundant when summaries are generated for each host involved in this example.

Several operations that extend beyond the scope of the original work have been included. These include *finger*, *telnet*, *ping*, network compression, *traceroute*, a modified *fping*, basic SNMP information, and a screen refresh. These options seem to contradict the notion of passive monitoring because they create traffic, but in practice these sorts of tools become useful when it becomes necessary to probe for network or host information. Other miscellaneous options such as protocol summaries, filtering, postscript dumps, and varying node and link time-outs are available via menu options and scroll bar adjustments.

```
Summary of Communications for host cujo.curtin.edu.au
    From cujo.curtin.edu.au to ubvmsb.cc.buffalo.edu
        TCP_SMTP pkts = 8 (0.56 K)
    From butler.acsu.buffalo.edu to cujo.curtin.edu.au
        UPD_DOMAIN pkts = 1 (0.08 K)
    From cujo.curtin.edu.au to uniwa.uwa.edu.au
        TCP_NNTP pkts = 489 (29.08 K)
    From uniwa.uwa.edu.au to cujo.curtin.edu.au
        TCP_NNTP pkts = 731 (349.04 K)
    From cujo.curtin.edu.au to wvnvaxa.wvnet.edu
        TCP_SMTP pkts = 9 (1.34 K)
        UPD_DOMAIN pkts = 1 (0.08 K)
    From cupid.curtin.edu.au to cujo.curtin.edu.au
        UPD_NTP pkts = 4 (0.35 K)
        TCP_NNTP pkts = 12 (0.73 K)
        TCP_LOGIN pkts = 98 (5.74 K)
    ...
```

Figure 6. Sample host protocol summary.

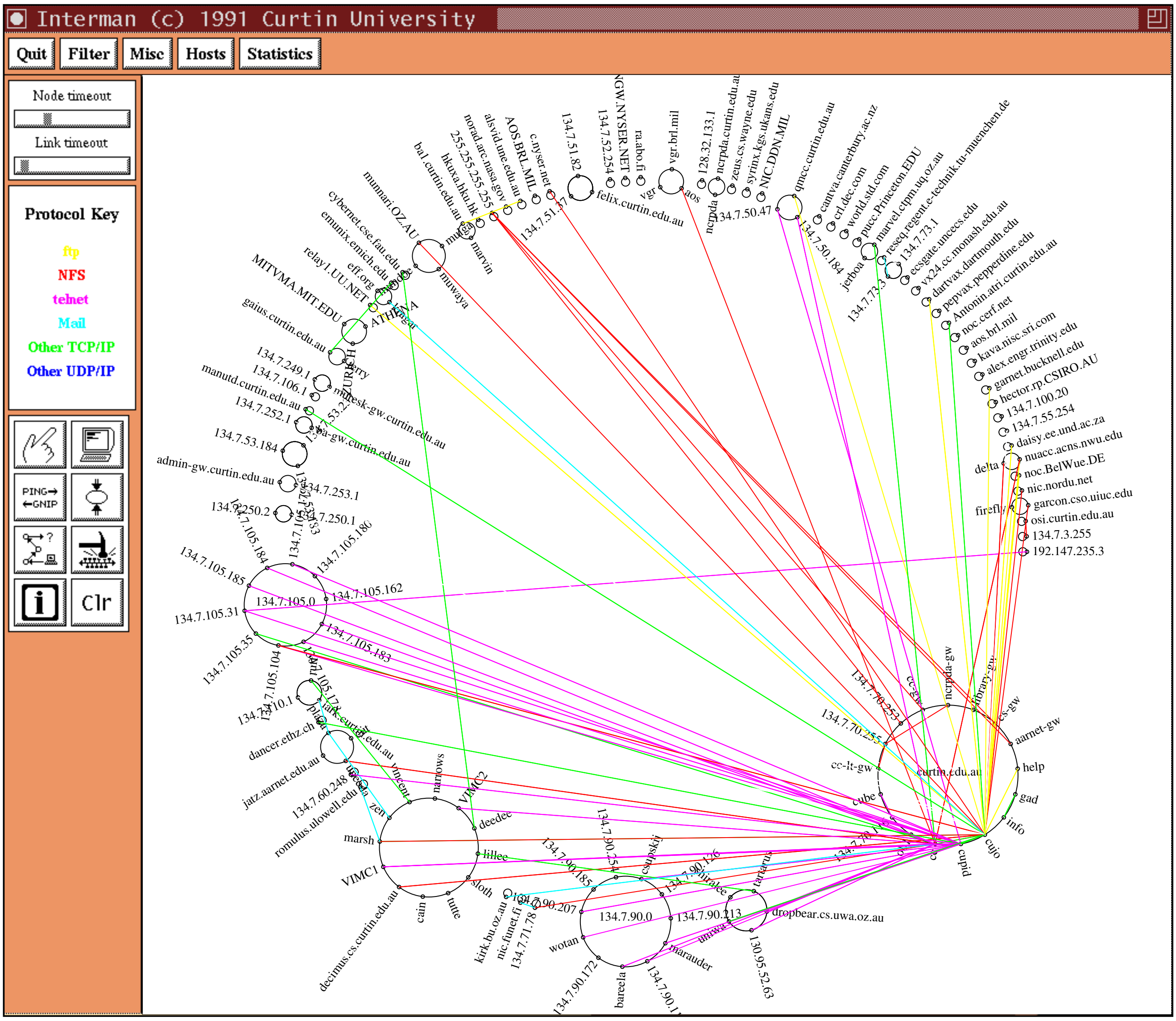

Figure 5. *interman* showing IP communications on the Curtin University campus backbone.

4.3 Packet Analysis; *packetman*

Branching away from the notion of real-time traffic analysis, we decided to implement a retrospective packet analyser. In practise, *packetman* can be used to decipher packet trains which are buffered, and optionally stored for future reference. A protocol analyser has the advantages of being able to decompose headers and examine protocol transactions in great depth. Our implementation, while far from complete, provides a good base from which we can work to provide comprehensive protocol analysis.

The display is segmented into three windows each providing a different view of captured network data. The top window is a sequential trace of captured data. Note that a filter may be applied before commencing a trace to allow the user to focus on transactions of interest. Each packet has an associated sequence number, timestamp, source/destination pair, and a brief overview of the protocol contained within (see Figure 7).

A textual description of all decodable protocol fields within the selected packet is displayed in the middle window. At present, decomposition for Ethernet frame types, selected IP/UDP and IP/TCP, ARP and ICMP protocols have been implemented. The lower window gives a simple hexadecimal and ASCII dump of the entire packet.

5 Future Directions

5.1 Enhancements to Existing Tools

With the addition of extra data structures and protocol analysis routines, we have found that a reduction in performance has resulted. Fortunately, we have several options available to help improve overall display performance and statistical accuracy.

A recent paper[19] has outlined a new packet capturing mechanism, BPF[18], that claims a performance

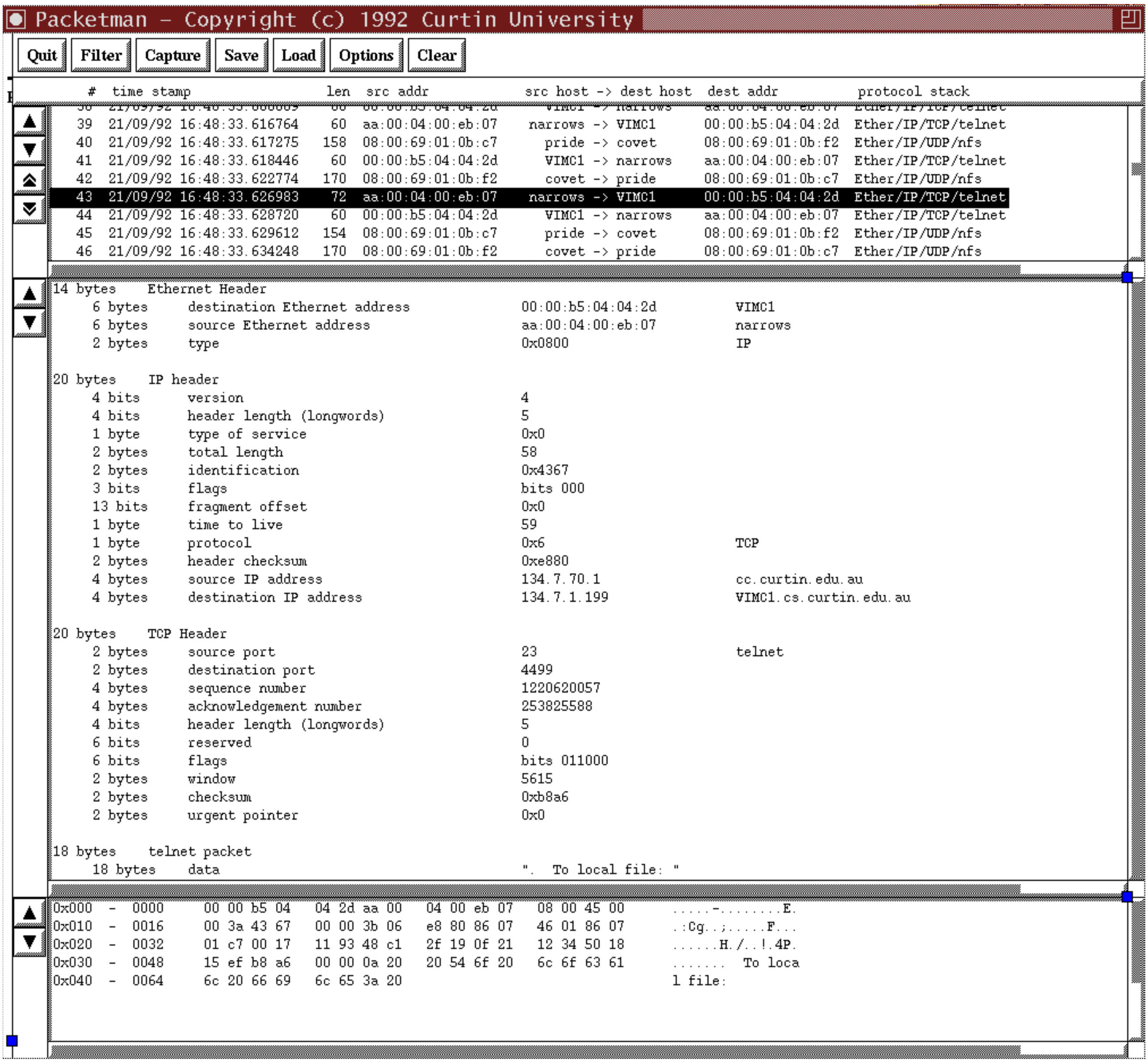

Figure 7. Using *packetman* to examine a telnet packet.

increase of 10 to 150 times over NIT. Although implementing code to support the BPF is trivial, we have no way of testing the implementations as we do not have access to the required SunOS kernel source. Unfortunately, the port of BPF for Ultrix and other common platforms is not currently available, but it is hoped that a mechanism will be incorporated at compile time to determine whether BPF support is available, otherwise the system standard network tap will be used. This will allow us to support as many platforms as possible in the same manner as *Tcpdump*.

A major contributor to poor performance is the sheer volume of X protocol transmissions. Although this is hardly surprising within a dynamic graphical application such as ours, we can reduce the number of X server requests to a fraction of what they are now by improving font handling within the X server. Enhancements such as these[6] have already been made available in the public domain as extensions to the X11R5 font server. Although such extensions are currently available, they are not implemented on a broad cross-section of X capable workstations. We are hoping that extensions to the X protocol, or a similar mechanism, will be included as part of the X11R6 distribution to provide this support. This will allow us to re-work our display routines and provide a reduced display update interval.

Calculation of traffic flow statistics also has room for improvement. At present, our flow calculations are inaccurate due to inter-chunk gaps being ignored. We have decided to implement the *augmented loadring* algorithm[24] to provide us with the correct flow statistics.

Packetman requires a great deal of work in terms of additional protocol modules. We will continue to expand the capabilities of the IP module as best we can, but cannot guarantee that other major protocols such as DECnet, Novell, EtherTalk etc. will be attempted[12].

5.2 SNMP Management Station

Implementing a useful network management station using SNMP will be the most challenging aspect of the project. We have not decided how the display will work, as the question of the ideal method of auto-topology remains unanswered[40]. We would like to implement an RMON[39] capable management station that will access the active host information and connection matrices of RMON probes. To date we are still concentrating our efforts on passive monitoring, but hope to enter the management scenario in the near future.

5.3 Representing Physical Networks

A project was initiated early in 1992 to investigate the possibilities of physical network representations. It was envisaged that this tool would be used to document physical network schemas with a minimum of effort on the user's part. This tool would use icons extracted from familiar vendor logos, and be able to interconnect these icons with lines representing Ethernet cable and fibre optic cable.

Although a very successful undergraduate project, in terms of X programming and network design experience, this tool requires a great deal more development in order to be usable. We hope to continue the development of this tool and also perhaps to, couple it with a monitoring engine to provide a crude auto-discovery capability.

5.4 Geographical Traceroute

Most people are familiar with the internet route tracing facility, *traceroute*[15]. Some people may also be aware of the uumap project which attempts to identify sites globally. If these two information resources were coupled with the CIA World Bank II[7] world map, it would be possible to produce a "geographical traceroute" which displays the geographical "point-to-point" routes taken to reach a remote site.

The information provided by such a tool could be used as either a diagnostic tool or as a educational aid for those who have not grasped the basics of IP routing. Other possibilities such as overlaying national and international backbones and satellite links could provide a more in-depth view of Internet connectivity. This project is currently under development.

6 Conclusion

We have presented an overview of a research project currently underway at Curtin University. At present we have implemented some utilities which we feel provide the network administrator with an immediate and intuitive understanding of network utilisation. We have successfully used these tools to diagnose and correct a variety of faults on several networks.

These tools were not designed to replace existing network management systems, rather to supplement them. We believe that by adding the information provided by these monitoring tools to the facilities provided by management tools like *SunNet Manager*[35] or *HP Open View*[11], the network administrator will receive a more complete and powerful management system.

7 Availability

Interman, *etherman*, and *packetman* are available via anonymous ftp from host `ftp.cs.curtin.edu.au` as a binary distribution for DECstations and SUN SPARCstations. Depending on which machine type you

12. If there is sufficient interest, we will consider releasing the source for public review.

require, the directories `pub/netman/sun4c` and `dec-mips` contain the files: `etherman.tar.Z`, `interman.tar.Z`, and `packetman.tar.Z`. At present, source code is not available.

8 Acknowledgements

Much of our work has been a distillation of some of the publicly available source code provided by the Internet community. Many thanks to Jeff Mogul, Steve McCanne, Van Jacobsen, Craig Leres, Dave Curry, Robert Braden and Annette DeSchon and a multitude of developers across the Internet for their contributions of utilities such as *Tcpdump*, *nfswatch*, *NNStat* etc. which helped inspire the project. Thanks to George Ferguson for his efforts in *xarchie* which guided the development of our GUIs. Special thanks go to Phil Dench for his original implementation of *etherman* under IRIX (which will one day return), and his willingness to help when *gl* was all too hard. Thanks also to Peter Elford and Geoff Huston for encouragement, testing and colour postscript dumps. To all the current users across the Internet community, thanks for bug reports, suggestions, kind words and your patience. Last of all, thanks to all members of staff within the department who contributed to this project in many ways.

References

[1] Boggs, D.R., Mogul, J.C., Kent, C.A. Measured Capacity of an Ethernet: Myths and Reality. In Proceedings of SIGCOMM '88 (Stanford, CA, Aug. 1988), ACM.

[2] Braden, R.T. A Pseudo-machine for Packet Monitoring and Statistics. In Proceedings of SIGCOMM '88 (Stanford, CA, Aug. 1988), ACM.

[3] Braden, R.T., DeSchon, A.L. *NNStat*: Internet Statistics Collection Package -- Introduction and Users Guide Release 2.3 edition, USC / Information Sciences Institute, Marina del Ray, CA, 1989.

[4] Case, J., Fedor, M., Schoffstall, M., Davin, J.A. Simple Network Management Protocol. Request for Comments #1157, Network Information Centre, SRI International, May 1990.

[5] Curry, D., Mogul, J. *nfswatch*(8l) manual page. Purdue University, IL, March 1993.

[6] Deininger, A., Meyers, N. Using the New Font Capabilities of HP-Donated Font server Enhancements. The X Resource, O'Reilly & Associates, Inc. CA, Issue 3, 1993

[7] Dellinger, J. CIA World Bank II map data and source code. Stanford University

[8] Digital Equipment Corporation. The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications (Version 1.0). Digital Equipment Corporation, Intel, Xerox, 1980.

[9] Digital Equipment Corporation. *packetfilter*(4), Ultrix V4.2 Manual.

[10] Gettys, J., Scheifler, R.W., Newman, R. *Xlib* C Language X Interface, X Version 11 Release 5. MIT X Consortium, MA, Aug.1991.

[11] Hewlett Packard. *HP Openview* reference manual.

[12] Hook, D.G. The *hershey*(3) manual page. Melbourne University, Melbourne, 1992.

[13] Hook, D.G. The *vogl*(4) manual page. Melbourne University, Melbourne, 1992

[14] Hull, C. The *EtherView*(1) manual page. University of Vermont, 1991.

[15] Jacobsen, V. The *traceroute*(8) manual page. Lawrence Berkeley Laboratory, Berkeley, CA, Feb. 1989.

[16] Jacobsen, V., Leres, C., and McCanne, S. The *Tcpdump*(1) manual page. Lawrence Berkeley Laboratory, Berkeley, CA, June 1989.

[17] Kiethley, K.S. Three-D Athena Widgets (*Xaw3d*) source code, Jet Propulsion Laboratories, NASA, Feb. 1993.

[18] McCanne, S. The BPF Manual page. Lawrence Berkeley Laboratory, Berkeley, CA, May 1991.

[19] McCanne, S., Jacobsen, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In 1993 Winter USENIX conference proceedings (San Diego, CA, Jan. 1993).

[20] McCloghrie, K., Rose, M. Management Information Base for Network Management of TCP/IP Based Internets: MIBII. Request for Comments #1213, Network Information Centre, SRI International, March 1991.

[21] McCormack, J., Asente, P., Swick, R.R. *X Toolkit* Intrinsics - C Language Interface, X Version 11 Release 5. Massachusetts Institute of Technology, MA, August 1991.

[22] Metcalf, R.M., Boggs, D.R. Ethernet: Distributed Packet Switching for Local Computer Networks. Communications of the ACM 19(7):395-404, July 1976.

[23] MIT X Consortium. X11 Windowing System Release 5. Massachusetts Institute of Technology, MA, Aug. 1991.

[24] Mogul, J. C. Efficient Use of Workstations for Passive Monitoring of Local Area Networks. In Proceedings of SIGCOMM '90 (Philadelphia, PA, Sept. 1990), ACM.

[25] Mogul, J. C., Rashid, R.F., and Accetta, M.J. The Packet Filter: An Efficient Mechanism for User-level Network Code. In Proceedings of 11th Symposium on Operating Systems Principles (Austin, TX, Nov. 1987), ACM, pp. 39-51

[26] Peterson, C.D. Athena Widget Set - C Language Interface, X Version 11, Release 5. Massachusetts Institute of Technology, MA, July 1991.

[27] Pochmara, J. *xtr* source code, Dec. 1989, Oregon Graduate Institute, Beaverton, OR

[28] Postel, J. Internet Protocol, Request for Comments #791, Network Information Centre, SRI International, September 1981.

[29] Silicon Graphics, Inc. NetVisualyzer reference manual, Silicon Graphics, Inc. Mountain View, CA.

[30] Silicon Graphics, Inc. *snoop*(6D) manual page. Silicon Graphics, Inc. Mountain View, CA, 1990.

[31] Silicon Graphics, Inc. Graphics Library reference manual, C edition (version 4.0). Silicon Graphics, Inc. Mountain View, CA, 1990.

[32] Sun Microsystems, Inc. *etherd*(8c); SunOS 4.1.1 Reference Manual. Mountain View, CA, Oct. 1990. Part Number: 800-5480-10.

[33] Sun Microsystems, Inc. *etherfind*(8c); SunOS 4.1.1 Reference Manual. Mountain View, CA, Oct. 1990. Part Number: 800-5480-10.

[34] Sun Microsystems, Inc. *NIT*(4P); SunOS 4.1.1 Reference Manual. Mountain View, CA, Oct. 1990. Part Number: 800-5480-10.

[35] Sun Microsystems, Inc. SunNet Manager 1.1 Installation and Users's Guide. Mountain View, CA, 1991.

[36] Sun Microsystems Inc. *traffic*(1c); SunOS 4.1.1 Reference Manual. Mountain View, CA, Oct. 1990. Part Number: 800-5480-10.

[37] Tremblay, J.P., Manohar, R. Discrete Mathematical Structures with Applications in Computer Science. McGraw-Hill 1975.

[38] Trewitt, G. M. Topological Analysis of Local-area Internetworks. In Proceedings of SIGCOMM '88 (Stanford, CA, Aug. 1988), ACM.

[39] Waldbusser, S. Remote Network Monitoring Management Information Base. Request for Comments #1271, Network Information Centre, SRI International, Nov. 1991.

[40] Waldbusser, S. Exposing the Myths about Autotopology. The Simple Times, C/o Dover Beach Consulting, Inc., CA, 1(1):5-6, March 1992.