Fremont: A System for Discovering Network Characteristics and Problems

David C. M. Wood, Sean S. Coleman, & Michael F. Schwartz – University of Colorado

ABSTRACT

In this paper we present an architecture and prototype implementation for discovering key network characteristics, such as hosts, gateways, and topology. The Fremont system uses an extensible set of modules to discover information, based on a variety of different protocols and information sources, rather than a single network management protocol. This approach allows more complete and timely information to be discovered than, for example, using only one protocol, even one as capable as the Simple Network Management Protocol. The discovered information is time-stamped and recorded in a database. The contents of this database are cross-correlated to form an increasingly complete network picture, to direct further discovery, and to highlight inconsistent information.

Introduction

The Scenario

Everything looked OK on the network monitor when your boss walked in, complaining that she couldn't get to the Ancient History server in the Classics department. Now you're in trouble. Everything you normally monitor is obviously up, but the problem just won't go away. But no problem, if you have the tool that will tell you what the route is supposed to be to get to the Classics subnet. You had heard before that they were on the network, but you never knew that the connection was via a Sun workstation / gateway in the Athletics department. After a quick call, you can report back to your boss that the coach has plugged his workstation back in, and the history server should be accessible in ten minutes.

Well, probably the campus network wouldn't really depend on careful administration in the Athletics department. Nonetheless, tracking changes and problems in a campus network is difficult, because authority and responsibility for various network segments is distributed across multiple organizations. Even on the segments that *are* well controlled, users (particularly those departing the institution) have no incentive to report that they are removing their host from the network. It is usually not an emergency, but it is useful to find out about such activities, particularly before one runs out of network addresses on a segment.

Motivating The Approach

"What's the big deal?" you ask. "I can use traceroute[9] to track down this routing problem." Perhaps. But traceroute really works best when the network is functioning properly. When there are problems, traceroute alone may not identify the problem. There may be multiple paths between a host and destination; over time the routes may change. Maybe you are experiencing a performance bottleneck, rather than a network partition. Observing that traffic passes across a subnet with a large number of hosts attached to it may help explain the problem. Other problems may also arise. For example, on any large network occasionally two hosts get configured with the same IP address. This generally makes communications impossible for either host. Detecting this problem is relatively easy if you have a tool that remembers the IP and Ethernet associations longer than the usual timeout of the ARP cache[15].

"Well," you continue, "I can use traceroute to find the path, and then I can use the Simple Network Management Protocol (SNMP)[2] to check the packet arrival rates at each gateway along the way. That handles the performance problem. For multiple network addresses I would..."

Sure, you can do it manually, but this is the sort of thing that computers are supposed do well. The fragmented nature of the existing network management tools makes life difficult.

There are two types of problems with current network management tools. First, each tool takes a particular perspective, and cannot support network management functions that require other perspectives. For example, SNMP treats the network as a set of instrumented devices. It can only retrieve information from nodes where SNMP agents are running, and cannot perform active probing tasks (such as traceroute). Moreover, since it focuses on managing known devices, it cannot aid in the discovery of devices or services, and the network manager must invest significant effort in configuring these tools.

More generally, different sources of network information have different characteristics with respect to timeliness of discovered information, discovery expense, danger of generating network problems (such as broadcast storms), and completeness of discovered information.

These differences lead to the second problem: network managers must manually cross-correlate information obtained from several tools. The tediously detailed nature of this information makes it virtually impossible for a network manager to harvest all of the useful information that is potentially available. Instead, when problems arise the manager probes the small window of information needed to solve that problem.

What is needed is a framework for network management that combines the various tools into an integrated system. The system should observe many different aspects of network state, and integrate the information into a coherent picture. Given such a picture, the network manager can learn of problems earlier, and can view a current picture of various aspects of the network more easily.

System Description

Overview

John Charles Fremont was a nineteenth century explorer in Colorado, California, and other parts of the western United States[4]. Inspired by his versatility, we named our network discovery system after Fremont. The Fremont system architecture is illustrated in Figure 1. In this figure, light lines indicate control flow, and heavy lines indicate data flow.

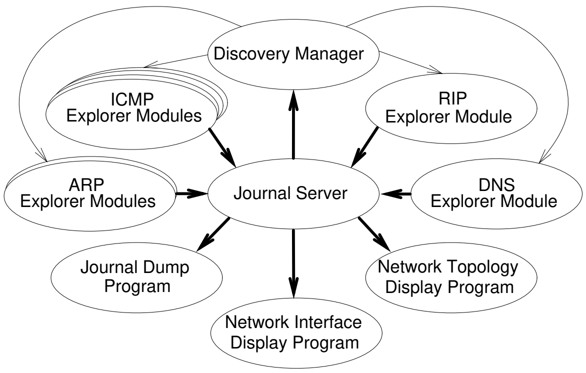

Figure 1: Fremont System Architecture

The Fremont system is based on an extensible suite of *Explorer Modules*, each of which uses a commonly available, existing network protocol or information source to uncover network information. This range of modules supports a broad set of discovery mechanisms and techniques. Some Explorer Modules actively probe the network, sending packets out into the network, and watching their effects. Other Explorer Modules generate no network traffic, and instead quietly observe the network activity around them. For example, passive packet monitoring allows routing information to be collected without imposing added processing load on a gateway. More active, directed probing allows both local and remote networks to be examined without the need for installing specialized network monitoring hardware or software.

Just as Fremont the explorer kept a dated journal of his activities, the Fremont system records discovered information in a central repository, which we call the *Journal*. This Journal is managed by the *Journal Server*, which serializes updates, timestamps and records the data, and answers queries from programs that wish to interrogate the Journal.

The activities of any good explorer are heavily influenced by experiences along the way. The Fremont system supports this function by way of a *Discovery Manager*. The Discovery Manager interrogates the Journal, and compares information discovered from the various Explorer Modules to determine a more complete picture of network characteristics (such as topology), and direct further discovery. Because every discovered feature is time stamped with its original date of discovery, its last change, and its last verification, network changes are easy to track. The Journal can also be interrogated by user interface agents, as will be discussed later in this paper.

The Fremont system is intended for a variety of network environments. Because all modules communicate via BSD sockets, there are no restrictions about the physical location of individual modules. Moreover, the system can be replicated at multiple sites, exploring different networks, and sharing information among the replicated components.

Explorer Modules

The current Fremont prototype supports 8 different Explorer Modules, based on 4 different information sources. For each information source, we give a brief description of the nature of the source, what type of information the Explorer Modules can discover using that source, and a detailed discussion of how each Explorer Modules operates. The detailed discussions include the conditions under which each information source can and cannot successfully discover network information, as well as how the information can be cross-correlated with other discovered information to provide insight into the represented networks.

The reader interested only in a high-level understanding of the Fremont system can read the first 2-3 paragraphs in each Explorer Module subsection, and skip the detailed discussions.

Address Resolution Protocol Explorer Modules

The Address Resolution Protocol (ARP) provides a mapping between Medium Access Control (MAC) and network layer addresses (e.g., between Ethernet and IP addresses)[15]. Whenever a host tries to send a packet to another host on a shared subnet, the sending host must first look up the Ethernet address in the local host's ARP table. If there is no entry for that IP address, the host must broadcast an ARP request for the destination Ethernet address.

The host that claims to have that IP address will reply to the requester.

Fremont has two Explorer Modules that discover and record the mappings provided by the Address Resolution Protocol. This information can be used in many cases to determine the manufacturer of the discovered interface[1]. It can also aid in determining gateways, locating changes in interface configurations, and discovering multiple interfaces with the same network layer address.

Fremont's ARPwatch Explorer Module passively monitors ARP message exchanges, and builds a table of Ethernet/IP address pairs for the directly attached subnets. Because this module uses the Network Interface Tap (NIT) feature of SunOS, this module must be run with system privileges.

Fremont also has an EtherHostProbe[12] Explorer Module, which attempts to send an IP packet to the UDP Echo port of each host in a range of addresses. Doing so causes the originating host to generate ARP requests, the responses for which are entered into the host's ARP table, and then read by the EtherHostProbe Explorer Module. For each address probed, one ARP request is broadcast. In addition, if there is a host on the network with the probed address, it will generate an ARP reply. The originating host will then send the UDP packet to the Echo port of the probed host, and the probed host will, if so configured, reply to that packet. In summary, there is an ARP request broadcast for each address probed, and then two or three additional packets will appear on the network for each responding host. The module limits the rate of generated packets to four per second. It does not use the Network Interface Tap and does not require special privileges.

Fremont has two ARP-based Explorer Modules because each module has different strengths and weaknesses. The ARPwatch module requires special privileges, and will not discover hosts that are not recipients of traffic from other hosts. This module generates no network traffic, and can be left to run for long periods of time. The EtherHostProbe module generates traffic, and does not require special privileges. It provides more thorough discovery, and will finish in an amount of time limited by the number of addresses probed.

Both modules share some common limitations. Both are limited to gathering information only about hosts that are on a directly attached, locally shared subnet (e.g., hosts on the same Ethernet as the one on which the Explorer Module is running). Both modules must ignore "proxy" ARP replies, where a gateway issues an ARP reply for hosts that are behind the gateway. This is easily done when the remote hosts are on a different subnet, but some gateway devices will reply for a set of addresses on the local subnet. Our solution in these cases is to recognize the device type when the multiple IP addresses are reported for a single Ethernet address. Except for this case, multiple IP addresses for a single Ethernet address usually indicates that a system has been reconfigured. Multiple Ethernet addresses for a single IP address usually indicates a misconfigured host, which is using the IP address assigned to some other host.

Internet Control Message Protocol Explorer Modules

The Internet Control Message Protocol (ICMP) provides a variety of network layer information, as part of the Internet Protocol (IP)[16]. Fremont has four Explorer Modules based on ICMP. Since ICMP packets are usually processed at high priority in router queues, the ICMP Explorer Modules take precautions not to send them too frequently.

The first two ICMP Explorer Modules are based on ICMP "Echo Request" and "Echo Reply" messages. These messages are used by the UNIX "ping" program, to test if a remote host is reachable[14]. The purpose of these two modules is to identify operational interfaces attached to the network. The first Explorer Module is based on sequential pings. This module sequences through a range of addresses, recording when it receives replies.

The second ICMP Explorer Module is based on broadcast pings. This module sends an ICMP Echo Request to the broadcast address of the subnet being probed. These *directed broadcasts* tend to be less successful than sequential pings on a subnet with many hosts, because closely spaced replies can cause many collisions. However, if used carefully, broadcast ping can be an effective interface discovery tool for large subnets of class A and B networks. In particular, it works well if the address space is large but there are not very many hosts on the individual subnets. In these cases, a sequential search of the address space would take a long time.

Directed broadcast packets that are broadcast with a time-to-live field larger than one can cause severe broadcast storms, due to incorrect networking software implementations or configurations in even just one host on a network. To minimize this problem, the broadcast ping Explorer Module sends packets with minimal time-to-live values (determined dynamically, in a fashion similar to the sequential increase mechanism used by traceroute; see below). To avoid broadcast storms, many gateways are configured not to broadcast packets that have a directed broadcast address as the destination address. This avoids the problem, but also reduces the effectiveness of the broadcast ping module.

[1]Throughout this paper we use the term *interface* to refer to a network interface, i.e., a separately addressable network connection to a machine.

The third ICMP Explorer Module is based on ICMP mask request/reply messages for determining the subnet mask of a network interface. This is not as widely implemented as the echo request/reply. In fact, some implementations allow the interface to be configured not to respond to subnet mask requests, to avoid problems with hosts configuring themselves based on incorrect subnet mask replies. Nonetheless, Fremont uses this feature of ICMP to discover and record the subnet masks of all the interfaces that it has already discovered. Fremont uses the collected subnet masts to aid in determining the network structure. It also uses the gathered information to detect conflicting subnet masks on different interfaces of a subnet.

The fourth ICMP Explorer Module uses a technique similar to Traceroute [9], to determine the route a packet would take from source to destination. Normally, when a packet is sent towards a destination, it has a "Time-To-Live" (TTL) field that is set large enough that the packet may reach its destination, yet small enough that undue network resources will not be consumed if the packet gets caught in a routing loop. As each router along the path receives the packet, it decrements the TTL field. If a router decrements the TTL field to zero, that router is supposed to drop the packet and send an ICMP Time Exceeded packet back to the source of the original packet. If the TTL is still non-zero, then the router forwards the packet on to the next router closer to the destination.

Traceroute takes advantage of this feature to determine routes by sending a sequence of UDP packets towards a destination host. The first packet in this sequence has a TTL of 1. That packet is dropped by the first router along the path, and that router sends an ICMP Time Exceeded message back to the source. The next packet is sent with a TTL of 2, and the second router along the path drops the packet and returns an ICMP Time Exceeded message. This process continues until the TTL of the packet sent is just large enough for the packet to reach its destination. The sequence of ICMP Time Exceeded messages returned to the source provides a trace of the routers along the path to the destination. The packets are sent to a port on the destination host that is unlikely to be used. Thus, when the packet arrives at the destination, it will typically cause the destination host to send either an ICMP Protocol Unreachable or ICMP Port Unreachable message.

Fremont's Traceroute Explorer Module uses this mechanism to determine the structure of the network surrounding the host on which the module is running. It does this by using the traceroute scheme to identify gateways and the subnets to which those gateways are connected.

Not all routers perform correctly as described above. Some hosts send their Unreachable message back to the source using the TTL field from the received packet, causing the packet not to arrive back at the source until the TTL of the original packet is large enough for an entire round trip. The Traceroute Explorer Module can handle most of the common failure modes, and hence provides an excellent means of discovering network topology.

The Traceroute Explorer Module sends packets towards three host addresses on the destination subnet, in an attempt to maximize the amount of information discovered. For example, if the module is sending packets towards subnet 128.138.238 (netmask of three bytes), then it sends packets to 128.138.238.0, 128.138.238.1, and 128.138.238.2. If a host receives a packet that is addressed to host zero on the subnet, the host is supposed to treat that packet as though it were addressed to that host. The module therefore sends to host zero on the destination subnet to maximize the chance of getting a reply from a node on that subnet. We send to two other addresses on that subnet, on the assumption that although one of those addresses may actually be the interface address of the gateway that accepted the packet addressed to host zero, the other address will not be that same gateway. The gateway should then send a final ICMP Time Exceeded message as it decrements the TTL to zero and drops the packet destined to some other address on the subnet.

The Traceroute module continues to send packets towards as yet unreached destinations while waiting to timeout packets it has sent to other destinations. It ensures that no more than eight packets per second appear on the network as a result of this parallel activity. With a ten second timeout for a response, this can result in up to 80 outstanding packets at any one time. However, most of these packets will have been lost somewhere, and the module will just be waiting for them to time out.

The Traceroute module is designed to operate on a local, high-speed network. Although the module will work across shared, low-speed serial connections, it stops tracing towards a particular destination if that trace reaches any of several national backbone networks.

Because it will receive ICMP Time Exceeded messages from only the single closest interface on the routers along the traced path, the Traceroute module will only discover half the interfaces traversed. Running this module from multiple locations in the network will acquire more complete information about the router interface addresses.

The current implementation of this module does not make any attempts to discover multiple paths, although the internal data structures are in place to accommodate this. Alternate routes can be discovered by running the module from different points in the network or, in some cases, simply by running it at different times. For example, if a lower priority, redundant path exists between two locations, that path will be discovered only when the primary path is down. Since this module, like all of the other Explorer Modules, stores its information in the Journal, the Journal will contain more complete information aggregated from multiple invocations of this module.

Routing Information Protocol Explorer Module

The Routing Information Protocol (RIP) uses broadcast messages to advertise routes to particular networks, subnets, or hosts[7]. Although it has fairly limited capabilities, RIP is widely used. Each RIP packet from a router contains a list of network addresses and metrics. No subnet mask information is contained in these packets, so routes to networks, subnets, or hosts are determined by comparing the subnet mask of the receiving host to the address being advertised. Subnet advertisements are not propagated outside of the network to which the subnets belong.

Unfortunately, not all RIP sources are trustworthy. Many badly configured hosts "promiscuously" rebroadcast all learned routing information without regard to the subnet from which that information was learned. This gives the false impression that the host may really have a separate route to the advertised networks. Fremont's RIP Explorer Module attempts to identify those RIP sources that appear to be operating in this erroneous manner.

The RIP module monitors RIP advertisements on shared subnets, building a list of hosts, subnets, and networks as they are seen in the advertisements. The collected data is recorded in the Journal, and used as clues for further discovery probes.

Like the ARPwatch module, the RIPwatch module uses the Sun NIT with a packet filter to watch the RIP packets on the shared subnets. This means that this module must run with system privileges, and that the module can see no further than the directly attached subnets.

Domain Naming System Explorer Module

The Domain Naming System (DNS) stores name, address, name server, and other information about interfaces in a distributed, hierarchical database[13]. Names and addresses are both stored in two distributed tree structures. One tree is organized to permit easy address lookups given a domain name. The other tree is organized to permit easy name lookups given an IP address. This latter tree is often called the "reverse" domain.

Fremont's DNS Explorer Module searches the appropriate subtree for all addresses in a specified network. The primary purpose of this module is to discover network topology by identifying gateways. This module was derived from the "nslookup" program, which is part of the Berkeley Internet Name Domain server distribution[10]. Nslookup understands how to format queries for all of the different types of data that the DNS supports, and how to interpret the results.

The DNS module retrieves the set of all address-to-name mappings from a domain, using "zone transfers". It does this by descending recursively into the DNS tree starting from a specific point, in a manner similar to Ganatra's Census program[6]. This technique creates no more network or name server load than is caused by a secondary DNS server.

The DNS module also uses ICMP Mask Requests to retrieve the subnet mask from one of the first hosts discovered on the desired network.[2] This is usually one of the name servers, thus increasing the likelihood that the returned mask is correct. Using the subnet mask and the information obtained from the DNS tree, the module tries to determine which sets of interfaces comprise gateways. It does this by looking for several different matches. The most obvious case is when multiple IP addresses correspond to the same machine name. The DNS module also looks for multiple names for the same address, and then looks for matches within those groups. It further looks for names which differ only by "-gw" or similar naming conventions.[3] This module also looks for "designated" gateways[13], but this does not appear to be a widely implemented use of the DNS.

The DNS module records in the Journal the number of hosts on each subnet and the highest and lowest addresses assigned on each subnet. Since the module has the complete set of name/address pairs for the network being examined, it could send all of this information to the Journal. However, because this information is readily available from the DNS, we do not record a name/address pair if it is the only information that we have involving an interface.

Journal

Each Explorer Module collects some subset of the available network data, depending on where the module runs and which protocol or information source it uses for discovery. This information is recorded in the Journal.

Most recorded data are used to provide a representation of the network. These data may then be used to answer user queries about the network entities and structure. In addition, some of the recorded data are used as a guide to further

[2] The DNS module was written before we had a Journal server. Since it needed the subnet mask in order to know how to allocate interfaces to subnets, and since we wanted to make this as automatic as possible, we chose to have the DNS module invoke the Subnet Mask module.

[3] In the future we will identify *possible* gateways using other weaker heuristics, tagging the resulting entries in the database with a "questionable quality" flag.

discovery. For example, the data collected from RIP packets provide strong indications about the existence of specific other networks and subnets. This information is used by the traceroute Explorer Module to improve its performance.

The Journal data are grouped into records representing interfaces, gateways, and subnets. Table 1 shows the primary fields that are maintained for interface records.

MAC layer address
Network layer address
DNS name
Subnet mask
Gateway to which this interface belongs

Table 1: Interface Fields

Gateways are represented as collections of interfaces. For gateways, we also record the subnets to which they are connected. The reason for doing this is that the Traceroute Explorer Module is able, in some cases, to determine the subnet to which a gateway is attached without being able to determine the address of the interface on that subnet.

For each discovered subnet, we record a list of gateways attached to that subnet. Note that there are cases where we may have discovered a subnet, but do not yet know what gateways are connected to that subnet.

All data items are stored with the date and time of initial discovery, last change, and last verification. This information is useful for observing several network characteristics. For example, we can see when hosts have been removed from the network. When this happens, Fremont stops updating the interface data record (except perhaps via the DNS Explorer Module). A network manager can observe this, and then contact the owner of the missing host to verify that the network address can be reused.

Because it is the shared place where observations are stored, and because there are several Explorer Modules recording complimentary findings there, the Journal is more that just the sum of its parts. For example, the fact that the same Ethernet address is observed by two ARP modules running on different subnets is not significant until that information is written into the Journal. Only then, because of the common storage, can that gateway be discovered. Similarly, both the Traceroute and DNS Explorer Modules collect information about gateways, and store that information in the Journal. Because the two modules use different techniques, the resulting data in the Journal are more complete than might be determined by either module acting alone.

Journal Server

The Journal Server maintains an in-memory representation of the Journal data, which it writes to disk periodically and at termination.

As noted in the Journal description, the stored data are grouped into records representing each interface, gateway, or subnet. Each of record is stored in a linked list for that type of data. The lists are ordered by time of last modification, so that the most recently changed items are at the end of the list. The data records for interfaces are indexed by three AVL trees, for lookups by Ethernet address, IP address, and DNS name. This allows quick access to individual data records, as well as access to ranges of records. An AVL tree is also used to index subnet records by subnet address. Gateways can be accessed by any one of their interfaces. The storage requirements are shown in Table 2.

Record	Bytes/Record
Interface	200
Gateway	84
Subnet	76

Table 2: Journal Storage Requirements

Note that in a distributed environment, no one Journal Server would need to store information about much more than the local network. Hence, storage requirements are modest. For example, a 25% full class B network (16k interfaces) with 192 subnets used (and an equal number of gateways) would require under four megabytes of memory.

The Journal Server responds to three primary requests: Store/Update, Get, and Delete. These requests are supported through a common library of access and data transfer routines that the Explorer Modules, Discovery Manager, and data analysis and presentation programs use. The Get function may return multiple data records depending on the selection criteria in the request.

Discovery Manager

The purpose of the Discovery Manager is to decide what information needs to be collected and what Explorer Modules should be invoked to collect those data. The Discovery Manager initializes itself by reading a startup/history file containing the address of the Journal Server, and the command name, invocation frequency, and information about recent runs for each Explorer Module.[4] It then opens a connection to the Journal Server and retrieves the data related to the attached subnets. It next adjusts the schedule for running any particular Explorer Module, based on the data already collected. The

[4] The startup/history file was implemented before we had a Journal server. In the future we will move this data to the Journal.

startup/history file records what each Explorer Module needs for input, and what features it discovers. The current list is shown in Table 3.

As the Discovery Manager runs the various Explorer Modules, it updates the startup/history file, which is used to determine what modules to run next. For example, if the Discovery Manager sees that 20 of 400 interfaces recorded in the Journal do not have subnet masks recorded and that this was true before the "subnet mask" module was last invoked, then the Discovery Manager will not shorten the interval until the next invocation of that module. This ensures that the resulting exploration effort is as fruitful as possible.

Info. Source	Module Name	Inputs	Outputs
ARP	ARP-watch	none	Enet. & IP addr. matches (over time)
	Ether-HostProbe	IP addrs.	Enet. & IP addr. matches (immediately)
ICMP	Sequential-Ping	IP addr. range	Intf. IP addrs.
	Broadcast-Ping	Subnets or Nets	Intf. IP addrs.
	Subnet-Masks	IP addrs.	Subnet Masks
	Traceroute	Subnets, Nets, or nothing	Intfs. per gateway; gateway-subnet links
RIP	RIP-watch	none	Subnets, Nets, Hosts
DNS	DNS	Network number	Intfs. per gateway

Table 3: Explorer Module Input/Output

When the Discovery Manager starts an Explorer Module, the Discovery Manager has several mechanisms for directing the Explorer Module. The particular mechanism for each Explorer Module is recorded in the Discovery Manager startup/history file. Most Explorer Modules, if given no specific direction, will examine the directly connected networks or subnets.

Presentation Programs

The ultimate purpose of Fremont is to provide some insight into the network being explored. To this end, we have built three programs for viewing the data available in the Journal. The first program simply lists all of the data in the Journal. We used this for early debugging.

The second program presents the interface data in three levels of detail, using X window displays. The first level lists all interfaces in a particular network, including the network layer address, DNS name, and time since last verification of existence (ignoring time of last DNS verification). This gives an easy indication of when the interface was last observed on the network. The second level lists all subnet interfaces, including the MAC layer address (if available), an indication of whether or not this is a source of RIP packets, and an indication of whether this is one interface of a gateway. The third level lists all of the data items stored in the Journal for a particular interface. This program is useful for looking at the state of the network interfaces over time. With it, a network manager can note which machines are out of service.

The third program provides an X window display of the network structure, as represented in the Journal. This is built from the gateway and subnet records stored in the Journal. The program retrieves the network and gateway entries from the Journal, and dumps the data in the format expected by SunNet Manager[24]. We then use SunNet Manager to display the data, as illustrated in Figure 2 (showing the output for a part of the University of Colorado network discovered by Fremont). This use of Fremont provides a significant improvement to SunNet Manager. While SunNet Manager provides a *discover tool* that checks the routing table on the local machine to find subnets, it does not uncover the relationships between the subnets. Using SunNet Manager, the user must enter and maintain network relationship information manually. Fremont supports this function automatically.

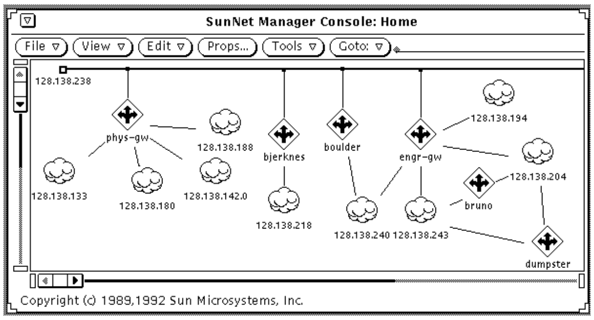

Figure 2: Discovering Subnets

Analysis Programs

We have two programs that analyze the Journal data to uncover possible network problems. The first lists subnet mask conflicts for all of the interfaces in the same network. With this information we can identify hosts that are not configured properly for a subnetted environment.

The second analysis program lists the possible conflicts between MAC layer and network layer addresses. In particular, we locate cases where multiple interface records have the same network layer address for different media access addresses, or vice versa. The first case represents either changing hardware or two different hosts using the same network layer address. The reverse situation may represent a system configuration change, a gateway doing proxy ARP, or the multiple interfaces of a gateway.

Evaluation And Experiences

In this section we present measurements and comparisons of the various Explorer Modules, to help evaluate the overall cost and effectiveness of the Fremont system.

Network and System Load

Table 4 shows the intervals that we have found appropriate for network discovery, the time required for completion of each invocation, and estimates of the network and module host system load resulting from that module.

Using the intervals specified, network and system load is kept reasonable. We have also installed precautions to ensure that Fremont will not adversely affect the network on which it is running. For example, the system stops tracing towards a particular destination if it detects a routing loop. Also, the modules that use parallel network activity to improve performance limit the rate at which packets are generated. The modules that use the Network Interface Tap to watch an attached subnet minimize the load on the host system by packet filtering.

Discovery Effectiveness

Table 5 shows the results of a brief run of Fremont, exploring one of the subnets in the Computer Science Department at the University of Colorado. For this run, all active modules were run once. Results for the one passive-monitoring module (ARPwatch) are given after the first 30 minutes of monitoring, as well as after 24 hours. As can be seen, quite a few interfaces were discovered almost immediately, and monitoring network traffic for a day caused most interfaces to be discovered.

To compute the "% of Total" column, we presume that the DNS data are an accurate reflection of the network. In the case of the network we tested this is a reasonable assumption, because the people who operate that subnet are very conscientious about keeping the DNS current. In fact, when we scrutinized the DNS records, we found only two entries for which there were no real machines connected to the network. From this we concluded that the DNS data showed slightly more interfaces than actually existed. We did not find any interfaces on the subnet that were not in the DNS.

Table 6 shows measurements of discovering the subnets of the campus network. We have assigned about 114 subnets, but several of those are not in use at this time. The RIPwatch module discovered 111 subnets. This can be treated as the exact number of subnets since, if we cannot find a route to a subnet on campus, then effectively it is not connected to the campus network. The DNS module found 93

Module	Min/Max Interval	Time to Complete	Network Load	System Load
ARPwatch	2 hours; 1 week	continuous	none	minimal
EtherHostProbe	1 day; 1 week	1 sec/address	1 - 4 pkts/sec	minimal
SeqPing	2 days; 2 weeks	2 sec/address	.5 pkts/sec	minimal
BrdcstPing	1 week; 4 weeks	30 sec/subnet	short storm	short high load
SubnetMasks	1 day; 1 week	2 sec/address	.5 pkts/sec	minimal
Traceroute	2 days; 2 weeks	5 - 20 minutes	4 - 8 pkts/sec	moderate
RIPWATCH	2 hours; 1 week	2 minutes	none	minimal
DNS	2 days; 2 weeks	1 - 5 minutes	10 pkts/sec	high

Table 4: Explorer Module Characteristics

Module	Interfaces	% of Total	Reason for loss
ARPwatch	34	61	Run for 30 min
	50	89	Run for 24 hours
EtherHostProbe	48	86	Not all hosts up when run
BrdcstPing	42	75	Collisions
SeqPing	38	70	Not all hosts up when run
DNS	56	100	Not necessarily current

Table 5: Discovering Interfaces on a Subnet Results from 1 Run of Each Active Module

subnets. This is because not all subnet managers enter their interface addresses into the name service. The DNS module further found 31 gateways connecting 48 of those subnets. Note that each of Tables 5 and 6 showed only the modules relevant to their discovery task (interfaces for Table 5, subnets for Table 6). Not all modules are used for both tasks.

Module	Subnets	% of Total	Comments
Traceroute	86	77	Gateway software Problems
RIPwatch	111	100	Nearly all subnets advertized
DNS	93	84	Not all hosts name served
DNS	48	43	Subnets with gateways identified

Table 6: Discovering Subnets Results from 1 Run of Each Active Module

Observations

In the following paragraphs, we offer observations from our experiences with the various Explorer Modules. In particular, we address such features as reliability, completeness, time to completion, and network and system resource consumption.

The Sequential Ping Explorer Module is the simplest and most reliable of the modules, because virtually every host implements the ICMP Echo Request/Reply protocol. The load presented to the network is low, because request packets are sent only once every two seconds. This will result in one reply packet for each existing host. If the module receives no response to a packet after issuing one request to each destination address, it sends one more request packet to each destination that did not respond. The second request rarely succeeds on a local network unless either the network or the remote host are heavily loaded. Running this on a class C network takes between 9 and 18 minutes. Running this on an entire class B network address space would take between 36 and 72 hours.

The Broadcast Ping Explorer Module presents a brief flood of ICMP Echo Reply packets. On a network with many hosts, this can provide a stress test of collision handling implementations, and usually results in lost packets, including both ICMP Echo Replies and normal traffic. Therefore, the reliability of this module suffers. The tradeoff is that this module completes in 20 seconds on a directly attached network.

On our networks, the ARPwatch and RIPwatch Explorer Modules consume minimal system resources on the hosts running those modules. Neither module generates any traffic, and both use the NIT to reduce the resource demands on the machine running those modules. Similarly, the EtherHostProbe and Subnet Mask modules offer only minor loads to the network and the machine running those modules. Like the Sequential Ping Explorer Module, these modules can take a long time to examine a large address space.

The Domain Name System Explorer Module operates in two phases. During the first phase, the module makes DNS requests of a name server for the network being examined. The network load is noticeable while the module does "zone transfers", as required to descend the DNS tree below the desired network. This activity takes about half of the time used by this module. During the second phase, the module searches the collected information for gateways. This is CPU intensive, particularly during the search for names with suffixes indicating possible gateways.

The Traceroute Explorer Module is modest in the demands that it places on either the network or the host system. This is mainly because we understood that traceroute activity might have significant impact on the network. We therefore were careful to impose limits on the load presented by this module. This conservative approach expands the time that it takes for this module to complete its exploration. We recommend that this module only be used to explore high speed networks (Ethernets or faster). The module has command line parameters that allow it to be slowed down more than the default value, as should normally be done when used on a slower network.

We did not implement an SNMP module in the current prototype because SNMP was running on only a few machines in our local internet when we started this project. Furthermore, SNMP requires knowledge of community names, which limits its ease of use. We plan to implement an SNMP Explorer Module (see the Future Work section).

In the DNS Explorer Module, we looked for *distinguished gateways*, as described in RFC 1035[13]. While this information could be useful for discerning network topology, we found that it is rarely supplied in the deployed DNS databases. Many other types of information are similarly unavailable or incorrect, such as lists of Well Known Services (WKS) and host and operating system type information.

Why is some information (like host names, addresses, and mail exchange records) available and reasonably up-to-date in the DNS, while other data is notoriously bad? The answer is that the data that *must* be correct in the Domain Naming System for proper operation in a networked environment generally *will* be correct. If a host system can function on the network without some particular piece of information being correct, current, or complete in the DNS, then it is quite likely that this information will be none of these. For example, the fact that a particular Well Known Service is running on a machine is more directly available in the distributed collection of /etc/inetd.conf files (which provide a list of the program locations for each service that is actually available on each machine). This is precisely why the WKS field in the DNS was deprecated in RFC 1123[1]. Network service information can also be determined by attempting to connect to a service, in the case of virtual-circuit based services[19]. Because of this, systems administrators rarely keep the WKS entries in the DNS up to date. These observations indicate that a name service works best for managing data needed for correct network operation, and that other types of data are better provided by a dynamic discovery process.

Related Work

A number of network management tools have been built using existing protocols[11, 17, 23]. However, these tools use only one or two sources of such information, and do not cross-correlate the data as Fremont does. Multiple information sources and existing protocols have been used to support resource discovery in other contexts as well, including Netfind (which discovers Internet user directory information)[20] and archie (which discovers files available via anonymous FTP)[5].

Robertson has built a system called *netdig*[18] that can discover network topology using SNMP. Several commercial network management stations also provide this capability. However, as with any use of SNMP, it is necessary to know the community name for every router in the network being examined. Most manufacturers' SNMP network management stations also offer some simple tools for drawing networks. The xnetdb program[3] does this at minimal cost, but it does no topology discovery, beyond connecting together hosts and gateways on the same subnet.

Future Work

We are currently extending Fremont to provide support for large internets, by caching data and supporting predicate-based queries to limit exchanged data to the parts that are needed. As a first step, we are making our software freely available, and encouraging people to set up Journal Servers throughout the Internet.

Another set of extensions in progress is adding Explorer Modules to use the two other protocols in their explorations. The first is SNMP. Although using SNMP requires knowledge of community strings, it is popular and powerful enough to allow improved topology discovery (as done by Columbia's *netdig* system). The second is Cisco Systems' Gateway Discovery Protocol (GDP). While not widely deployed, supporting GDP would help fill in some of Fremont's discovery gaps. A "promiscuous" mode network traffic monitor would be able to discover all communicating machines in a network. We will use this to extend our system into the discovery of network services.

We are also expanding our work with existing protocols. For example, beyond monitoring RIP advertisements, we plan to use directed probes to discover routing information, via the RIP Request and RIP Poll queries. The major advantage of doing so is that these requests and replies can be routed through a network, thus providing access to routing information on subnets other than just the local subnet. A problem, however, is that not all routers use RIP or respond properly to RIP Request or RIP Poll queries. Nevertheless, we expect to be able to identify some routers, and even some alternate paths using RIP queries [8].

Another area of Future work involves running the Traceroute Explorer Module from multiple points in a network. This is easy to do manually now, but will require remote execution capabilities in the Discovery Manager. We also plan to use "loose source routing", to look for multiple paths in the network. This feature of IP can allow the module to specify an intermediate router through which the traced path will be routed. We also plan to branch further from the local network, while continuing to minimize network impact. For example, if the network to be traced is only reachable through node G, and if G is exactly and always (for the duration of the traceroute run) H hops away from the host running the Traceroute module, then all traces can start with a TTL of H+1 rather than 1, because every packet will follow the same path from for the first H hops, and there is no need to continually re-trace the initial H hop path.

The data recorded in the Journal need to incorporate a more flexible notion of information *quality*. Currently we treat information discovered by some protocols as being of *better quality* than that discovered by other protocols. For example, data gathered using the ARP protocol are generally timely and correct, whereas DNS data are older and often subject to data entry errors. Thus, if the only indication of the existence of an interface is its record in the DNS, we would not add it to the Journal unless it appears to be part of a gateway. Similarly, we would like to have a flag to prevent continually retrying discovery of some datum that we know is unavailable. This would be similar to the negative caching concept that has been suggested for the DNS, in which the absence of an entry in the DNS could be locally cached to avoid unnecessary expense of future failed queries.

We plan further to examine the feasibility of extending the discovery processes to other protocols, particularly DECnet and OSI.

Our initial user inquiry agents focus primarily on rudimentary debugging tools with few graphical capabilities. We would like to have several tools that could provide real-time observation of the explorations and the discovered information, and a graphical visualization of the structure of the network as it is discovered. In the future, a more sophisticated interface could be integrated, perhaps from one of the commercial network management packages.

A final area of future work involves extending Fremont's graphical display mechanism to support dynamic updates, as new information is discovered.

Summary

The complexity of modern data communication networks has led to a situation in which network administrators must use a number of different tools to track changes and uncover problems in their networks. Because of no one tool provides a complete picture of the network, network managers have been forced to cross-correlate information obtained from several tools manually, often losing important information among the details.

The Fremont system provides a framework for network management that can combine many different tools into an integrated system. This approach represents an extension to the traditional network management paradigm, which treats the network only as a collection of instrumented devices (as in the case of SNMP). In addition to this paradigm, our approach supports passive traffic monitoring, active network probes, and information gleaned by cross-correlating data discovered from multiple sources. Because of this cross-correlation, Fremont provides more complete and useful information than any single network management system. It can also flag potential network problems based on inconsistencies in the discovered data. Fremont performs these functions without undue consumption of network or host system resources.

Table 7 summarizes the network characteristics that the current prototype discovers, based on the 8 different Explorer Modules we have implemented. This information is sufficient to provide detailed network maps, including topology maps (as illustrated in Figure 2), and tables showing the names and addresses of each host on each network, the local gateways used by each host, etc. In the future we expect to add route discovery capabilities to Fremont, at which time routing maps could also be produced.

Interfaces	Ethernet Address IP Address Name Subnet Mask Gateway Membership
Gateways	Interfaces on GW Subnets connected (topology)
Subnets	Gateways on Subnet (topology)

Table 7: Characteristics Discovered by Prototype

Table 8 summarizes the network problems that Fremont uncovers. The uncovered information can help network administrators solve a number of problems, such as those discussed in the Introduction section of this paper.

IP Addresses No Longer in Use
Hardware Changes
Inconsistent Network Masks
Duplicate Address Assignments
Promiscuous RIP Hosts

Table 8: Problems Uncovered by Prototype

In summary, the Fremont system provides an integrated framework for assisting a network manager in discovering network characteristics and trouble-shooting network problems. Because it makes use of many different information sources and network management tools, Fremont can form a more complete network picture than any one tool.

Prototype Software Availability

The Fremont software is available by anonymous FTP from ftp.cs.colorado.edu, in the directory pub/cs/distribs/fremont.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under grant NCR-9105372, and a grant from Sun Microsystems' Collaborative Research Program.

We thank Barb Dyker, Darren Hardy, Susan Smith, and the USENIX program committee for their helpful comments on this paper.

The Fremont system is based in part on the preliminary architecture described in[21]. This work is part of the Networked Resource Discovery Project at the University of Colorado[22].

References

1. R. Braden, ''Requirements for Internet Hosts – Application and Support,'' Internet Request For Comments 1123, October 1989.

2. J. D. Case, M. Fedor, M. L. Schoffstall & C. Davin, ''Simple Network Management Protocol,'' Internet Request For Comments 1157, May 1990.
3. H. Clark, *Xnetdb Software*, Ohio State University, June 1990. Available by anonymous FTP from thor.oar.net: /pub/xnetdb
4. F. Egan, *Fremont, Explorer for a Restless Nation,* Doubleday, Garden City, New York, 1977.
5. A. Emtage & P. Deutsch, ''Archie – An Electronic Directory Service for the Internet,'' *Proceedings of the USENIX Winter Conference*, pp. 93-110, San Francisco, California, January 1992.
6. N. K. Ganatra, *Census Software*, Department of Computer Science, University of California, Santa Cruz, June 1992. Available by anonymous FTP from ftp.cse.ucsc.edu: pub/csl/census.tar.Z
7. C. Hedrick, ''Routing Information Protocol,'' Internet Request For Comments 1058, Rutgers University, June 1988.
8. J. C. Honig, *RIPQUERY Software*, Cornell Theory Center, Cornell University, August 1991. Available as part of the gated distribution, by anonymous FTP from gated.cornell.edu: pub/gated/gated-2.1.tar.Z.
9. V. Jacobsen, *Traceroute Software*, Lawrence Berkeley Laboratories, December 1988. Available by anonymous FTP from ftp.ee.lbl.gov: pub/traceroute.tar.Z
10. M. Karels & J. Wood, *Nslookup Software*, University of California, Berkeley, September 1990. Available as part of the BIND distribution, by anonymous FTP from okeeffe.cs.berkeley.edu: /4.3/bind.4.8.3.tar.Z
11. K. Kislitzin, ''Network Monitoring by Scripts,'' *Lisa IV*, October 1990.
12. P. E. McKenney, *EtherHostProbe Software*, SRI International, July 1988. Available by anonymous FTP from phloem.uoregon.edu: /pub/src/etherhostprobe/etherhostprobe.tar.Z
13. P. Mockapetris, ''Domain Names – Implementation and Specification,'' Internet Request For Comments 1035, University of Southern California Information Sciences Institute, November 1987.
14. M. Muuss, *Ping Software*, U. S. Army Ballistic Research Laboratory, December 1983. Available by anonymous FTP from uunet.uu.net: /bsd_sources/src/ping
15. D. C. Plummer, ''An Ethernet Address Resolution Protocol – Or – Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware,'' Internet Request For Comments 826, November 1982.
16. J. Postel, ''Internet Control Message Protocol,'' Internet Request For Comments 792, University of Southern California Information Sciences Institute, September 1981.
17. W. C. Reissig, ''Dynamic Network Management Using the Simple Network Management Protocol (SNMP),'' Technical Report 90-08-04, Computer Science Department, University of Washington, Seattle, Washington, 1990. M.S. Thesis
18. S. Robertson, *Netdig Software*, Center for Telecommunications Research, Columbia University, August 1991. Available by anonymous FTP from ftp.ctr.columbia.edu: /pub/net/netdig.3.5.shar.Z
19. M. F. Schwartz, ''A Measurement Study of Changes in Service-Level Reachability in the Global TCP/IP Internet: Goals, Experimental Design, Implementation, and Policy Considerations,'' Internet Request For Comments 1273, November 1991.
20. M. F. Schwartz & P. G. Tsirigotis, ''Experience with a Semantically Cognizant Internet White Pages Directory Tool,'' *Journal of Internetworking: Research and Experience*, vol. 2, no. 1, pp. 23-50, March 1991.
21. M. F. Schwartz, D. H. Goldstein, R. K. Neves & D. C. M. Wood, ''An Architecture for Discovering and Visualizing Characteristics of Large Internets,'' Technical Report CU-CS-520-91, Department of Computer Science, University of Colorado, Boulder, Colorado, February 1991.
22. M. F. Schwartz, ''Internet Resource Discovery at the University of Colorado,'' To appear, *IEEE Computer Magazine*, Revised October 1992.
23. ''FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices,'' Internet Request For Comments 1147, SPARTA, Inc., April 1990.
24. *SunNet Manager 1.1 Installation and User's Guide,* Sun Microsystems, Inc., 1991.

Author Information

David Wood holds a B.S. in Electrical Engineering from the Massachusetts Institute of Technology and a M.S. in Electrical Engineering from the University of Colorado. He is the manager of wide-area and campus-wide networking at the University of Colorado. He is also a Ph.D. student at the University of Colorado. He can be reached at Computing and Network Services, University of Colorado, 3645 Marine Street, Boulder, CO 80309-0455, or via electronic mail at dcmwood@spot.colorado.edu.

Michael Schwartz received his PhD in Computer Science from the University of Washington.

He is currently an Assistant Professor of Computer Science at the University of Colorado - Boulder. His research focuses on issues raised by international networks and distributed systems, with particular focus on resource discovery and network measurement. Schwartz chairs an Internet Research Task Force Research Group on Resource Discovery and Directory Service, and is on the editorial boards for *IEEE/ACM Transactions on Networking* and for *Internet Society News*. He can be reached at the Computer Science Department, University of Colorado, Boulder, CO 80309-0430, or via electronic mail at schwartz@cs.colorado.edu.

Sean Coleman received his B.S. in Engineering Physics at the University of Colorado. He is currently an M.S. student in Computer Science at the University of Colorado. He is also a system administrator for a network of Suns. He can be reached at the Computer Science Department, University of Colorado, Boulder, CO 80309-0430, or electronically at coleman@cs.colorado.edu.