How to Keep Track of Your Network Configuration

J. Schonwalder & H. Langendorfer
TU Braunschweig, Germany

How to Keep Track of Your Network Configuration

J. Schönwälder & H. Langendörfer – TU Braunschweig, Germany

ABSTRACT

In this paper we present extensions for the *Ined* network editor allowing us to discover the structure of an IP network automatically. The discovering algorithm is based on an active probing technique that fits well with our interactive editor. We have chosen a multi-threaded approach to minimize response times which seems reasonable in fast networks but may fail when run over slow serial links. A set of utilities have been designed to lay out the discovered network based on additional information found in the Domain Name System.

Introduction

Maps showing the network configuration are very important documents for a variety of tasks. This motivated us to design and implement a domain specific editor called *Ined* that is specialized for drawing and maintaining network maps [7].

Ined provides all necessary editing features to draw and maintain network maps. Maps may be composed of node, network and link objects. Link objects represent connections between two node objects or a node and a network object. A large map can be structured into a hierarchical map using group objects that can contain other *Ined* objects. The editor has a programming interface allowing scripts written in the Tool Command Language (tcl) [3] to control objects shown inside of the editor. This interface enables users to write macros for frequently used command sequences. It also serves as an interface for scripts performing network management tasks.

Once we had a proper drawing program, we found it a very lengthy and tedious task to enter all the network information into the editor. Another problem is to keep network maps current since there are frequent changes in our network with about 1.300 registered hosts. So we decided to design and implement extensions for the *Ined* editor that will assist us in these tasks. The result of our work is an IP network discovery tool and a set of utilities simplifying the final layout process.

Design Decisions

When designing a network discovering program, you have to make some very important design decisions. First you have to determine the network layer on which to focus. It is much easier to design a smart discovering program if you have a clear understanding of what information should be discovered. We focus on the IP layer since this is the most frequently used protocol in our environment and it is supported by nearly every device on the network.

Next you have to decide if you will use passive monitoring techniques or if you will actively probe the network to gather information about its internal structure. The first alternative provides no direct way to guide the discovering process since you can only observe the current packet exchanges. Therefore it may take a long time gathering enough information to derive a complete picture of your network. Another problem with passive techniques is that you must have monitoring devices attached to all your subnetworks (e.g., Ethernets) and that the implementation under the UNIX operating systems requires a network interface tap which is not available on every machine.

The main advantage of passive monitoring techniques is that they put no additional load on the network itself. Using active probing techniques you must be aware of the additional load your network must carry. If your network is sensitive in this respect you will probably be in favor of passive techniques. On the other hand, active techniques can be directed easily. Therefore it is possible to minimize the time needed to get a complete map of your network.

Since our program should support an interactive editor we have decided to use an active probing technique. We are taking an aggressive approach in order to minimize the total time needed to discover a complete network. This is reasonable on fast networks but may fail when run over slow serial links.

The third design decision concerns the information sources available to the discovering program. There are two realistic choices. You can either use a network management protocol (e.g., SNMP [12]) to gather information, or you can try to use features of protocols handling ordinary network traffic for your purpose.[1] Using a management protocol seems attractive at first sight but it requires that most

[1] Sure, you can invent your own protocol. But it is very unlikely that it will be installed on every machine in a large network.

machines on your network are able to respond to management protocol requests. The second alternative is more likely to work with a large set of different devices but makes the analysis of the acquired data difficult.

The discovering algorithm described in the next section mainly uses the ICMP protocol [5] to discover the structure of a network. This has the advantage that every device on the IP network can be detected since every IP implementation must support the ICMP protocol. Additional information is retrieved from the Domain Name System (DNS) if available.

How Discovering Works

The network discovering algorithm is divided into nine steps. The first four steps send probing packets to gather data while the remaining steps are used for data analysis. The algorithm assumes an implementation which will be able to send request packets in a round-robin fashion while waiting for outstanding reply packets to arrive or timeout. This way the total time needed for a given address space remains constant, regardless of the number of responding hosts.

1. Determine IP addresses in use by sending ICMP echo request packets to every address of the given address space. We use a sequential approach since directed broadcast ICMP echo requests tend to cause lots of collisions or even broadcast storms due to errors in some IP implementations. Some IP router even remove incoming broadcast packets reducing the usefulness of directed broadcasts.
2. Trace the routes to all IP addresses discovered in step one using the Van Jacobsen algorithm [10]. The traces are stored for later analysis. Gateways showing up during the traces are added to the IP address list.
3. Determine the network mask for each IP address using the ICMP mask request. The network masks are saved for later analysis.
4. For every IP address, send an UDP packet to an unused port and save the IP address contained in the port unreachable reply packet. Some multi-homed devices respond to an incoming packet addressed to one of the remote interfaces with a packet containing the IP address of the incoming interface. The returned address is stored for later analysis.
5. Identify networks and subnetworks. Class A, B and C networks are easily recognized by examining the IP addresses. Subnetworks are a bit tricky. First collect all potential subnets based on the netmasks returned in step 3. Afterwards, check if the majority of all members of a potential subnet has reported a suitable netmask. This two level approach is needed to handle incorrect netmasks properly.
6. Identify multi-homed machines based on the traces and the address contained in the port unreachable reply packet of step four. Comparing the Domain Names of the IP addresses gives additional hints to multi-homed machines since the Domain Name Service often contains records with the same name for each interface of a multi-homed host.
7. Connect IP addresses to the networks identified during step six. Gateways are connected based on the traces. We found that gateways often return a different IP address when being traced. Therefore we skip the last hop of traces that end at a gateway machine.
8. Merge the IP addresses of multi-homed machines. This step cleans up duplicate information stored for each interface of a multi-homed machine.
9. Download the current map from the *Ined* editor and add all discovered objects to the map that do not exist yet. Hosts and gateways are mapped to node objects, networks to network objects and IP interfaces to link objects.

The above algorithm can be used to discover routing traces by initializing the list of IP addresses and starting the algorithm at step two. This is a very nice extension to the traceroute program [10]. The resulting map lets you easily identify machines where branches join that are important for your site.

Layout

The output of the discovering algorithm is a set of nodes, networks and links placed on a grid. Some separate tools help us to lay out the discovered topology. The separation of these tools from the discover process allows us to use them even when entering network maps interactively.

The layout tools are divided in a set of commands that are specific to IP networks and a set of commands that are of general use. The second set includes commands to manipulate the selection of the editor and the text shown inside the labels.

The IP specific commands mostly deal with the Domain Name Service (DNS). They fill empty attributes of *Ined* objects with DNS information or set icons automatically based on HINFO records. A table of regular expressions maps HINFO records to icon names.[2] Another IP specific command removes identical domain name endings from host names. This command shortens labels considerably resulting in more readable maps.

The most important layout utility rearranges icons on the map. We have chosen a simple method that places hosts connected to a network on a grid

[2] Many site administrators have their own naming convention although there are some guidelines and Official Machine Names in the Assigned Numbers RFC [9].

around the network. Hosts with more than one interface are placed on the border of a cluster. Another command allows us to group all nodes of a network having exactly one interface. Used in conjunction with the previous command, we get a map focusing on networks and their gateways.

The map in Figure 1 shows the networks owned by the computer science departments of our University. All 6 subnetworks are linked by the backbone network (134.169), the cisco router and the serial line network (134.169.247). Subnet 134.169.33 is shown as an expanded group. The icons have been arranged around the network and the domain name endings have been removed.

Implementation

The network discovering algorithm is implemented as a script written in the Tool Command Language (tcl). We have extended the standard tcl interpreter with a command allowing us to send ICMP packets. The use of a script language has the advantage to allow modifications and customizations at very low cost.

The ICMP module is implemented as a separate process called ntping. ntping runs setuid root since sending and receiving ICMP packets requires access to raw sockets. ntping creates a new job for each destination address read from stdin and processes these jobs in round-robin fashion. A tcl command hides the communication with ntping from the tcl programmer. Figure 2 shows the options of the tcl icmp command.

```
icmp [-size <b>] [-delay  <ms>]
     [-retries <n>] [-timeout <s>]
     [-ttl <n>] [-trace <n>] [-mask]
     <hosts>
```

Figure 2: tcl icmp command

The -ttl and -trace options are used to send an UDP packet to an unused port with a time to live set to <n>. The difference between these two options is that -trace always returns the IP address of the destination host while -ttl returns the IP address actually contained in the reply packet. The -delay option can be used to force a minimum delay between two successive packets. Delays have proven useful to reduce load on intermediate nodes like bridges or routers. The implementation of the delay option uses a loop with calls to gettimeofday for short intervalls since the usleep library call does not handle short intervals reliable on many machines.[3]

[3]Berkeley Unix uses signals to implement usleep. This causes a series of system calls which usually take more time to complete than the requested interval if the interval is short. The minimal delay obtained using usleep on a SparcStation was 20ms.

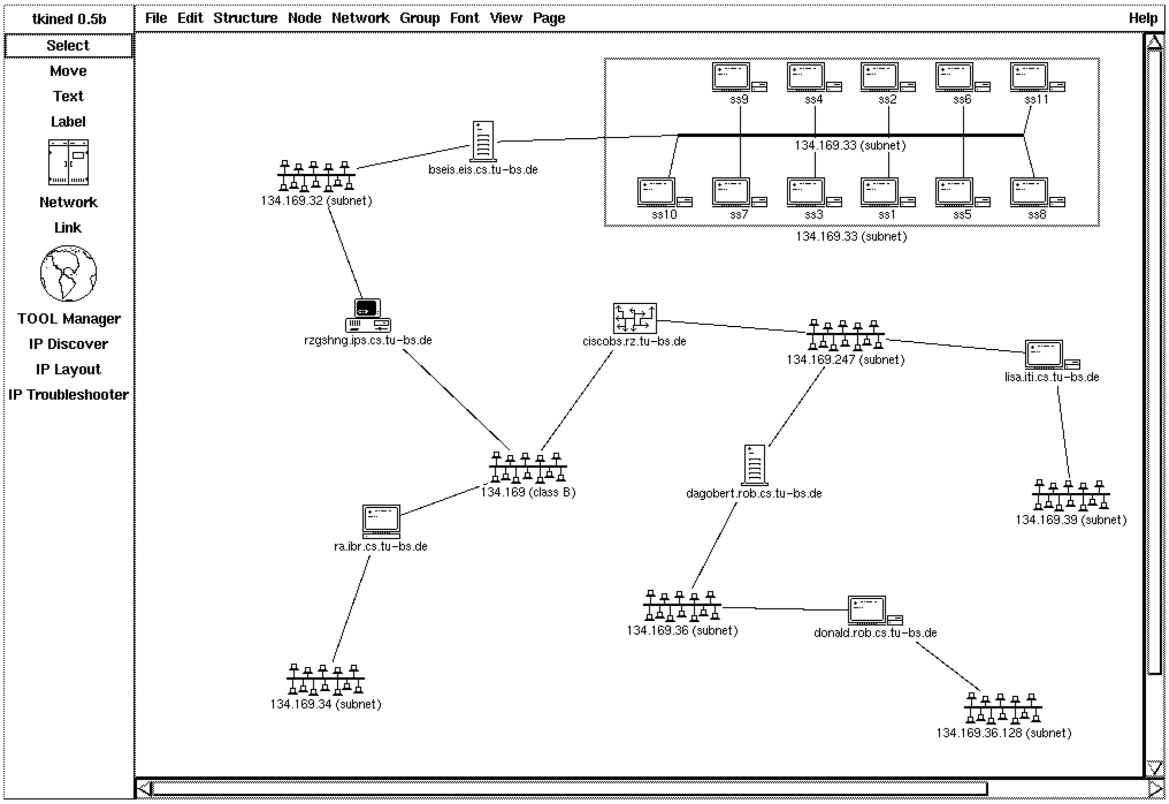

Figure 1: A sample Map

The efficiency of the script can be controlled by adjusting the parameters shown in table 1. We found the default values appropriate on our network (mainly Ethernet networks). You should increase the delay and timeout times if you have slow serial lines in your environment. The maximal length of a route only affects step two of our algorithm. Increasing this value is useful if you want to use the script to discover routes on a wide area network.

Parameter	Default
number of retries	3
timeout [s]	3
delay [ms]	5
max route length	8

Table 1: Discover Parameter

Table 2 shows some measurements for class C and class B networks. For each step of our algorithm, the number of discovered devices and the completion time is given. The most time consuming part is the initial pass to discover IP addresses in use. The current implementation breaks a class B address space in 255 class C like networks and processes these networks sequentially. Removing this limitation will increase the number of active threads which will reduce the overall time needed for class B networks.

Related Work

Another network discovering system called Fremont has been build by Wood, Coleman and Schwartz [8]. Fremont uses eight different explorer modules to discover network characteristics. All information gathered by these explorer modules is stored in a journal that is used to direct the exploring process and to uncover network problems.

The first difference of our approach and the Fremont systems is that we do not use passive monitoring techniques. Passive monitoring of network traffic only extracts information about the subnet the monitoring host is directly attached and it requires a long observation period. Since we are trying to discover network characteristics fast, passive monitoring techniques did not seem attractive to us.

The Fremont systems takes care not to load the network with probing packets. This seems reasonable since the explorer of the Fremont system makes regular passes through the network. Our system is meant to operate in fast local area networks, and is designed as a network analysis utility to be run occasionally. Hence an active probing algorithm seems applicable.

We further believe that gathering of information about the structure of subnetwork layers is best done by the hardware that connects these subnetworks (e.g., bridges and routers). Since more and more of these devices are capable of running SNMP, we are convinced that supporting SNMP and ICMP suffices.

Future Work

Our current work on the discovering tool will add support for SNMP. This will allow us to detect network configuration below the IP layer by querying hosts, gateways and bridges for their address translation tables. P. H. Kamp [1] has done some work to implement SNMP extensions for tcl supporting an asynchronous interface to send and receive SNMP packets. We expect to get the SNMP discover tools as fast as ntping using this asynchronous interface.

Routing tables retrieved via SNMP can be used to discover routes between two remote hosts. We will implement an algorithm similar to that of xpath [11] once we have finished our SNMP implementation.

Another activity is the integration of a configuration database in order to store information about discovered devices. The database is comparable to the journal server of the Fremont system but

Step	Class C		Class B	
	Number	Time [s]	Number	Time[s]
ICMP echo request	51	10	610	2453
Trace routes	51	9	610	289
ICMP mask request	52	4	610	24
Get remote interface address	52	3	610	15
Identify networks	2	2	5	32
Identify multi-homed machines	1	1	23	7
Connect hosts to networks	97	4	1004	260
Merge multi-homed machines	1	0	23	1
Create *Ined* objects	107	8	1219	102
Time to complete		41		3181

Table 2: Speed of the Discovering Script

may be used for general system management tasks as well [6].

Conclusions

In this paper we have presented a network discovering tool and how it provides an efficient method to get a map of your current IP network setup. The discovering mechanism is designed to operate on fast local area networks that wont suffer from the aggressive probing algorithm used. First experiments show that the approach is working fast and reliable. As noted by Wood, Coleman and Schwartz, discovering IP networks using ICMP packets is a very attractive approach. Other protocols that can be used for exploration are less widespread and they often require additional knowledge (like community names for SNMP).

Acknowledgements

We would like to thank Erik Schönfelder for the implementation of ntping. Also many thanks to Stefan Petri for his helpful experiments with directed broadcast pings.

Availability

The *Ined* editor has been re-implemented using the tk toolkit [4] and is now called tkined. It is available by anonymous ftp from ftp.ibr.cs.tu-bs.de in the directory /pub/local. You will find the source of our tcl interpreter scotty which includes ntping and the script described in this paper in the same directory. You are invited to join the tkined mailing list. Send your request message to tkined-request@ibr.cs.tu-bs.de

Author Information

Jürgen Schönwälder received his diploma degree in Computer Science from the Technical University of Braunschweig. He is now a member of the research staff at the Institute for Operating Systems and Computer Networks. His interests include system administration, network management, distributed systems and network security. Reach him via Mail at TU Braunschweig, Bültenweg 74/75, D-38106 Braunschweig, Germany. Electronic Mail should be sent to schoenw@ibr.cs.tu-bs.de.

Horst Langendörfer is Professor at the Technical University of Braunschweig since 1981. His research interests include operating systems, distributed systems, performance analysis, computer networks, network management and network security.

References

[1] P. H. Kamp. *tcl_snmp – SNMP interface for tool command language.* March, 1993.

[2] R. Lehman, G. Carpenter, and N. Hien, *Concurrent Network Management with a Distributed Management Tool.* Proc. of LISA VI, pages 235-244, 1992.

[3] J. K. Ousterhout. *TCL: An Embeddable Command Language.* Proc. Winter USENIX Conference, pages 133-146, 1990.

[4] J. K. Ousterhout. *An X11 Toolkit Based on the TCL Language.* Proc. Winter USENIX Conference, pages 105-115, 1991.

[5] J. Postel. *Internet Control Message Protocol.* RFC 792, September 1981.

[6] J. Schönwälder and H. Langendörfer. *Administration of large distributed UNIX LANs with BONES.* Proc. SANS II, Arlington, April 1993.

[7] J. Schönwälder and H. Langendörfer. *INED – An Application Independent Network Editor.* Proc. SANS II, Arlington, April 1993.

[8] D. C. M. Wood, S. S. Coleman, and M. F. Schwartz. *Fremont: A System for Discovering Network Characteristics and Problems.* Proc. USENIX Winter Conference, pages 335-347, January 1993.

[9] J. Reynolds and J. Postel. *Assigned Numbers.* RFC 1340, July 1992

[10] V. Jacobsen. *Traceroute Software.* Lawrence Berkeley Laboratories, December, 1988.

[11] A. Leinwand and J. Okamoto. *Two Network Management Tools.* Proc. Winter USENIX Conference, pages 195-205, 1990.

[12] J. Case, K. McCloghrie, M. Rose, S. Waldbusser. *Introduction to version 2 of the Internet-standard Network Management Framework.* RFC 1441, April 1993