Diagnosing Network Problems with Standard Tools

Jonas Lindberg & Erika Rice

December 10, 2004

Abstract

It is altogether too common that a user cannot use the network they are on to do the tasks they desire. Experts know how to diagnose network problems using tools available, but using these tools is hard for the non-expert. We have developed a tool that demonstrates the feasibility of automating the use of existing network tools.

1 Introduction

It is altogether too common that a user cannot use the network they are on to do the tasks they desire. When such a problem occurs, the user who is knowledgeable about networks will know the tools to use to pinpoint the problem. However, the average user, even the average computer savvy user, will not have this sort of expert knowledge at their disposal when the network acts up.

There are many tools that experts use to diagnose network problems. However, these tools can be difficult to use. They often have many options, and they place no interpretation on their output. It is up to the user to know what questions to ask and how to interpret what the answer means.

We have developed a prototype tool to examine the feasibility of developing an easy to use utility for diagnosing network problems. This tool, Why, requires only the website the user wants to connect to and a port to diagnose the network problem. The output is a description of what the tool thinks the problem is or an "I don't know" message with hints of the possible problems if it cannot determine the exact problem. Because this is a prototype tool, the focus was more on automating detection rather than creating a good user interface.

The rest of this paper is organized as follows. Section 2 describes the goals of this project. Next section 3 describes our general approach. This is followed by section 4 which describes the primary problems we are trying to diagnose. Section 5 gives the details of our implementation, including our diagnostic structure and the tools we use. Section 6 describes the tests we performed and discusses the individual test results. This is followed by section 7 which discusses some of the successes and challenges of this work. Sections 8 and 9 discuss, respectively, future and related work. Finally, we conclude in section 10.

2 Goals

This project has two evaluation goals. The first is to evaluate how well we demonstrate the feasibility of using existing tools to create a utility that can be used to determine the source of network problems with minimal expert knowledge needed by the user. The second goal is to test how well the tool we have developed diagnoses the specific problems we are trying to identify.

3 Approach

Our approach was to develop a tool that the user could invoke with a minimal number of arguments when she experienced network problems. Why invokes existing network tools and interprets the output so that the user can be given a concise summary of what the problem likely is. The primary technical

issue is determining the right tools to use and how to interpret their output. An expert diagnosing a network has the advantage of knowing the characteristics of that network; they know what is normal behavior and what is strange behavior. However, our goal was to make a light weight tool that could give some idea of the network problems without prior knowledge of what was normal for the network. This makes the task of knowing how to interpret the output from various network tools difficult.

4 Problem Areas

Because our goal is to test the feasibility of developing a general network diagnostic tool, we choose to focus on a fairly small number of common problems. These are mostly problems at one of the end hosts. Characterizing problems within the network is a problem that is beyond the scope of this project and developing tools that can do this is an area of on going research. To determine which problems it would be most useful to diagnose, we looked at several online network troubleshooting guides. We chose to focus on the problems which appeared to be most common.

4.1 Problems at the Local Host

4.1.1 Nameserver not accessible or invalid host

A source of connection problems is an inability to translate from a host name to an IP address. There are two potential sources of this problem. Either the local machine cannot contact any nameserver or the host does not exist.

4.1.2 No physical path

Suppose there is no physical path between one host and the host it is trying to connect to. The missing connection may be because the user has forgotten to attach their computer to the network (in which case it is easily fixable) or the missing connection may be somewhere outside of the user's control.

4.1.3 Improperly configured interfaces

Sometimes a user cannot connect to the network because there is no interface on the local machine configured to access the network. There may also be multiple external interfaces configured, and this may cause problems on the sending machine.

4.1.4 Local routing table incorrect

The local routing table on a host indicates the default gateway and interface to send packets on. It can also be used to specify alternate routes for specific destinations. For example, the local routing table could indicate packets destined for a particular destination be routed through another machine. If the network has been configured to use such a setup and the alternate router was unreachable, the user might not realize why most packets get through, but packets to a particular destination do not. We address only a subset of this problem wherein the default routes for all hosts are incorrect.

4.2 Problems at the Remote Host

4.2.1 Firewall

A badly configured firewall may block packets from reaching certain applications. Problems like this can occur when the server administrator accidentally enables a built-in firewall (e.g. by installing a service pack) or forgets to configure the firewall when installing a new service[1]. In other cases, a firewall may intentionally be blocking certain applications. In both cases, it would be beneficial to the user to know that a firewall is the reason they cannot reach the destination.

4.2.2 Service down

Services such as web servers do occasionally go down. It may be useful for a user to know that it is only the particular service that is down. For example, in the case where a user has a choice between an FTP download and an HTTP download. If HTTP is not

[1] A local firewall could also cause problems, but we are not directly considering that situation.

working, the user can use Why to find this out and use FTP instead.

4.2.3 Server offline

Sometimes a server is turned off, rebooting, or just not attached to the network. Just being able to detect that the computer is not on the network would be useful information for the end-user.

4.3 Internal Network Problems

Problems which occur within the network rather than at one of the end points are the hardest to diagnose. Problems such as congestion, routing problems due to routing around a link that is down, or inefficiencies due to routing between ISPs can be hard to pinpoint. As such, our only goal with respect to these problems is to be able to detect that the problem is somewhere within the network rather than at one of the end points. We do not see this as a terrible trade off, since it is problems at an end point that the user is likely to have some ability to remedy.

5 Implementation

5.1 Diagnostic Structure

Figures 1 and 2 show how Why diagnoses network problems. The first figure shows local tests and the second figure shows non-local tests; these tests are performed if all of the local tests pass. The rest of this section explains this process in detail.

The diagnosing process starts by trying to detect errors at or near the local host[2]. These tests are first because they are relatively quick; their running time does not significantly effect the time it takes for running the full diagnosis. The following tests are performed for identifying local problems: verifying the existence of a suitable network interface; verifying that name resolution works; and verifying that the given host name is valid. The last two tests are bypassed if the remote host is given as IP-address instead of host name.

If no local problems were detected, the process probes the remote host to detect non-local problems. First, Why tries to connect to the remote host on the given port. There are three possible outcomes from this test: port is open, port is closed or no response. A port-open response indicates that the host is reachable and that there in fact is a service listening on the port. Should Why instead receive a port-closed response, it concludes that the host is reachable but the service is down; the close, in this case, is an active response from the server saying that the port is closed[3].

The third case, not receiving any response at all, occurs when a firewall is blocking the port or the host is unreachable. Differentiating these two problems can be hard or even impossible. Our approach is to probe the host in different ways and hope that it will respond on some of our requests and prove itself reachable, which would implicate a firewall as the problem source. This technique will not provide us with any new information in the case where a firewall is blocking all incoming connection attempts.

If Why gets to the point where there does not appear to be any problem that it can diagnose, it tries a rough test for congestion. It does this by pinging the destination host for a short period of time; if any packets are dropped, congestion is suggested as the problem. Why also compares minimum and average round trip times for a set of pings to look for indications of congestion; if the minimum deviates too far from the average, congestion is reported as a potential problem.

5.2 Platform and Tools

Why has been developed for Linux. Development was done on Fedora Core 2 machines running 2.6 kernels and Debian Woody machines running 2.4 kernels. Our tool is implemented in Perl 5. The utilities used to implement Why are Traceroute (v. 1.4a12) [8], Nmap (v. 3.5 and 3.7) [3], Netstat (v. 1.42) [20],

[2]In our implementation, tests are performed sequentially. This is, of course, not as time efficient as running the tests in parallel, but performs well enough for this prototype.

[3]Although a firewall could give an active closed message for an open port, this is rare. Thus, we assume that a closed message means the port is really closed.

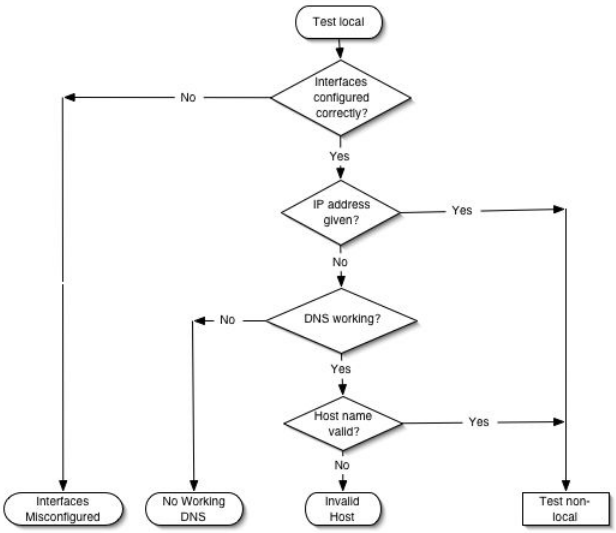

Figure 1: Flowchart illustrating the diagnosing process for local problems.

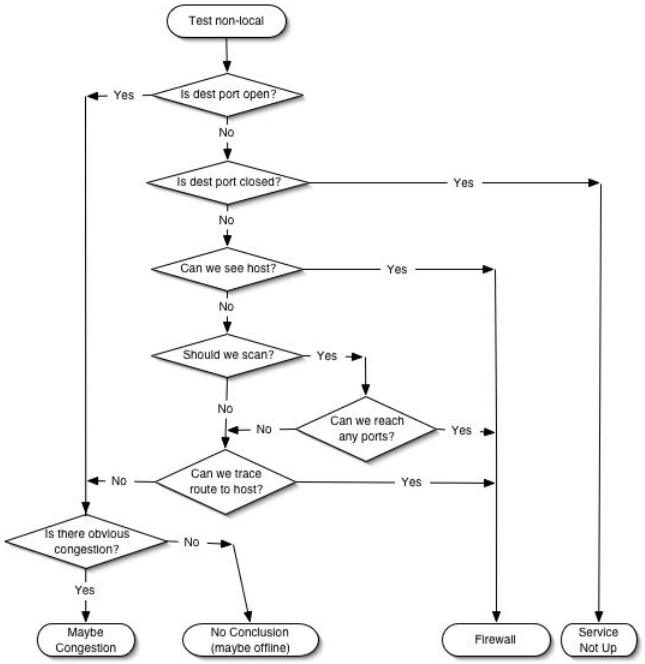

Figure 2: Flowchart illustrating the diagnosing process for non-local problems.

Ping [13], and Host [6]. These are all common Linux network tools.

The output of most tools is parsed by hand in our application. The one exception is Traceroute; there is a Perl module for parsing Traceroute output [5]. Such modules were not publicly available for other tools that we used. This presents one limitation of our current implementation of Why. The tool is dependent on the specific output formats of these tools; if the format of the output changes, our tool will break. A more modular design could address this limitation.

5.3 Usage

An easy to use diagnostic tool should take as few inputs as possible. Why requires two arguments: a host to connect to (either a host name or IP) and a port. An optional output granularity value controls the amount of information given to the user.

6 Testing

6.1 Unit Testing

Due to time constraints, the only systematic tests of Why we were able to perform were tests to detect each problem individually. Most problems occur at one of the end hosts and can easily be tested directly. Because we had access to multiple Linux machines, causing these problems was easy. However, because we only had root access on machines on a local area network of four computers, some tests could only be performed within that LAN rather than across the Internet.

6.1.1 Nameserver not accessible

In Linux, the list of local nameservers is kept in a file editable by the root user. This presents two ways of removing accessibility to the nameserver without having access to it. The first way is to delete the nameservers in the file. The second way is to put incorrect nameservers in the list. We did each of these in turn and tried to connect to `www.google.com` on port 80 (HTTP). In both cases, Why correctly reported that there were no accessible nameservers.

6.1.2 Invalid host

Performing this test was as simple as attempting to use the tool to diagnose the connection to a host which does not exist. The host and port used for the test were `theperfectroutingprotocol.com` and port 80. Why successfully tells us that this host does not exist. One could imagine improving the output by having Why do simple tests like changing the suffix (try `.net` instead of `.com`, etc.) and suggesting possible alternatives to the user.

6.1.3 No physical path

Why can only diagnose this problem when the break in the physical connection is at the local host. To test this, we disconnected the ethernet cable from a computer while leaving the ethernet interface configured. As done in most tests, Why was then used to try to diagnose the connection between the local host and `www.google.com` on port 80. When Why was run, it reported that it could not reach the default gateway and gave as possible reasons a broken physical connection, an incorrect gateway, or a malfunctioning gateway.

6.1.4 Improperly configured interfaces

Why was tested for this problem in two ways. In the first test, all interfaces to the network were disabled. Diagnosing with the standard `www.google.com` on port 80 gave the error that the only interface running was a local loop-back interface. Thus, the problem was diagnosed successfully.

The second test involved configuring two interfaces, an ethernet interface and a wireless interface, at the same time. Using the same parameters as before, this test reported that there were multiple interfaces configured. While technically correct, this answer is not terribly informative. Depending on how the interfaces affect the routing table, it may or may not be permissible to have multiple interfaces. If the host Why was being used on was, for example, a server where the two interfaces had been properly configured, Why would always print out this message as a probable source of error and would rarely be right. The proper way to detect if multiple interfaces are

problematic would be to combine the inspection of which interfaces are up with an inspection of the routing table. This information could be used to see if there was ambiguity in determining which interface packets to a particular destination should use.

6.1.5 Local routing table incorrect

The Linux utility Route [15] can be used to change the local routing tables on a host. Using Route it is possible to change the default gateway for all destinations or just the gateway to a particular subnet of destinations. The first test performed was to completely disable the default gateway. The result of trying to connect to `www.google.com` on port 80 was an error message stating that the gateway could not be reached. However, there was no indication that there was no gateway specified in the routing table.

The second test was to change the default gateway to be some IP that could not be directly reached from the local host. When run with the standard inputs, Why gave the same message as before indicating that the gateway could not be reached. However, in this case such a message is acceptable because a correct gateway that is down and an inaccessible gateway should look indistinguishable. However, it might be possible to give some indication that the gateway is wrong by checking whether or not it is on the same subnet as the local host.

6.1.6 Firewall

The Linux utility IPtables [7] can be used for setting up a firewall on a Linux machine. Using IPtables, all incoming connections to port 80 were blocked on a `meikon.homeip.net`. We then used Why to diagnose a connection on port 80 to that server from another machine. Why was able to suggest that the problem was a server side firewall blocking the connection.

As a second test, IPtables was used to block outgoing connections from the *local machine* to port 80 on other machines. When Why was used to diagnose the connection to `meikon.homeip.net` (without port 80 on the server blocked), the same message was given as before. This shows that our method of detecting firewall blocking can be used for either the local or remote host. However, since Why does not try to differentiate the two, a user of Why might be confused as to which side the firewall was on.

6.1.7 Service down

To test this problem, the Apache web server on `meikon.homeip.net` was taken down. Why was then used to diagnose a connection to that machine on port 80. The tool successfully reported that the problem was that the desired web service was down.

6.1.8 Server offline

This test was performed simply by disconnecting a host, `leela.cs.washington.edu`, from the network and diagnosing the connection to that machine on port 22 (SSH). The result of this test was that Why gave the message that the host was probably offline, but could not give any definitive results since the problem could also be a destination side firewall which drops all incoming connections.

6.1.9 Congestion

To test the capabilities of Why in detecting congestion, we set up NistNet [11] on a machine in a four computer local area network. NistNet is a kernel module that can be used on a Linux machine set up as a router. It monitors traffic going through the router and follows user defined rules on source destination pairs to delay packets, drop packets, and restrict bandwidth. Using this tool, we were able to emulate congestion.

The first test was to use the emulator to drop 5% of the packets between the test machine and the destination server, `leela.cs.washington.edu`. When Why was used to diagnose the connection to this machine on port 22, the output successfully classified the problem as congestion. This demonstrates that Why can correctly diagnose congestion that is heavy enough to cause packet loss.

The second test was to use the emulator to delay packets between the two test machines by 100ms (the normal delay between the two test machines is between 15 and 20ms). Why was not able to notice

congestion as the problem in this case and reported that it could not see any network problems. This is because our heuristic looks for variation in delay, but NistNet delayed packets uniformly. From this, we can conclude that Why can be used to detect congestion in some extreme cases, but cannot detect anything less than that.

6.2 Live testing

Throughout the development of Why there were times when we would experience network problems. During those times, Why was tried as a method of finding the source of the problem. The results of these spontaneous live tests are recorded here.

One morning, the network connection of one of the authors was not present. Why was used, and reported that the DNS was unavailable. A second use of Why was tried, giving it a known IP address instead of a host name. This time, the firewall blocking message was given. Hand testing confirmed that neither of these was the correct answer. However, the author was also unable to determine the problem.

In another situation, one of the authors used Why to successfully discover that the local area network was congested. Upon asking other users of the LAN, it was discovered that someone had been updating their computer and, therefore, generating large amounts of traffic on the local network.

Finally, while trying to find documentation for IPtables, one of the authors used Why to discover that the webserver was down. Hand testing confirmed that this was, indeed, the case.

Although these anecdotes do not count as rigorous testing, they do show that Why does have some utility for diagnosing real problems.

7 Discussion

In this section, we will discuss how our work meets the two evaluation goals previously described. We start with interpreting our test results and continue with discussing the implication our results have on the feasibility of constructing a network diagnosing tool.

7.1 Testing

To perform realistic testing on a real network diagnosing tool would be both hard and very time consuming. Not only would one need to know what problems can occur, how frequent each problem is, and how to diagnose problems in different hardware and software environments, one would also need to know how to simulate a broad range of problems and look at issues such as how the user interprets the output.

We have simplified this by concentrating on a small number of problems and by performing our tests in an isolated way. Most problems were easy to simulate; for the ones which were not, emulation was used.

In our tests, Why gave a technically correctly diagnosis for six of the nine problems. The problems that were not fully diagnosed were local routing table incorrect, host offline and congestion. In the first problem the diagnosis failed because we did not realize the possibility of a gateway that does not forward packets when designing the test. The host offline test failed in the sense that it could not determine, but merely guess, the source of the problem. However, we believe this is as good as can be done for this problem, since it can be impossible to differentiate from some firewall problems. Congestion was correctly diagnosed when packets were lost, but Why failed to infer the network congestion problem when significant packet delay was applied. The reason for this was that our heuristics where not sensitive enough. Finding generally applicable heuristics that works in heterogeneous environments is a difficult task. The best remedy for this is probably to construct the tool so that it can learn the characteristics of the environment it is used in.

Another interesting result from our test was that the diagnosis needs to be more detailed in order to avoid ambiguous results like those experienced in the tests of the firewall and the improperly configured interfaces problems.

7.2 Feasibility

We think that the tool we have developed shows that constructing tools for automatic network diagnosis is feasible. Why is far from a complete tool, but even

with limited diagnosing capabilities, it proved to be a useful tool in spontaneous live test. Why also diagnosed most problems in our systematic tests and provided useful hints on some of the other problems. However, in order for Why to be useful to the average user, many improvements would be required (see section 8).

Given more resources and time, it would be feasible to construct a tool that can diagnose or give educated guesses on many common connectivity problems. Such a tool could be very useful for the user who does not possess enough knowledge to efficiently diagnose network problems by hand.

8 Future Work

Although our goal was to demonstrate the feasibility of an easy to use network diagnostic tool, improving the user interface is still an important area of future work. In the context of this project, "easy to use" only means that Why takes minimal information and prints out error messages which indicate the source of the network problem. However, there are many improvements that could be made in the user interface. Currently Why takes a host and a port. This input could be improved by letting the user specify either a port or a type of service (e.g. 'web' or 'ssh'); this would let less experienced users use the tool with less effort. Further improvements could be made by making the output better organized and more informative.

The user experience would be improved if Why was faster. Currently a full diagnosis including port scans takes about 10 minutes. This time could be reduced by running several tests in parallel.

Of course, Why could always be improved by adding more diagnostic capabilities. There are several ways this could be done. One method of improving our diagnostic capability would be to increase the accuracy of the diagnoses we already do. Another method of improving the diagnostic capabilities would be to find more problems, determine how to solve them using existing tools, and integrate them into Why.

One area where there are many advances yet to be made is correct classification of problems within the network. This would involve finding better ways of detecting not only what problems within the network are (e.g., congestion, misdirected packets, etc.) but where those problems occur. Much work has been done on this problem [1, 8, 9], but there is still a long way to go before these tools become common.

Another area of improvement would address the problem of knowing what heuristics to use to determine when the network is acting abnormally. Currently, Why uses heuristics chosen by the authors based on their experience and general rules of thumb used by network administrators. The performance of Why would improve if it could be modified to constantly monitor the system and learn what is normal. Instead of *running* the program when network problems were suspected, the user would simply *query* the already running program.

Why does not take advantage of the extra knowledge that could be acquired by running as root. For example, some tools like Netstat and even Ping, have options that can only be fully utilized as root. There are other tools, such as Tcpdump [17] which can only be run as the root user. If the tool took advantage of these capabilities, it would be able to better discover the state of the network. If this were combined with the previous improvement, we could retain our original goal of not requiring special privileges by having the tool be run by the root user but query-able by normal users.

Currently, Why manually parses the output of most of the utilities it uses making the program highly sensitive to changes in the format of the output. This makes it harder to port Why to different systems and makes the whole program fail if future upgrades to those programs change the format of the output or the command line options of these tools. To decrease Why's dependence on these factors, it would be useful to break out calling the utilities and parsing their output into separate modules, in the style of the Traceroute module we used. Doing this would allow Why to remain relatively static in the face of changes. The only necessary changes would be changes to the modules to allow them to handle the new versions.

9 Related Work

There are many tools which have been developed for diagnosing Internet connectivity. These include the tools we used such as Ping, Traceroute, Nmap, Netstat, and Host as well as tools like Tcpdump (a packet monitoring tool), Pathchar (bandwidth characterization), and Tulip (a congestion characterization tool) [13, 8, 3, 20, 6, 17, 9, 1]. These tools put a wealth of information at the user's fingertips. However, none of them meet the requirement of being easy to use.

There has also been a large amount of work on systems for making large networks easier to diagnose by network administrators, some of which can be seen in [16]. Other work has focused on determine why large network enterprises fail [12]. However, neither these tools nor the information gained in these studies address the needs or the information gathering capability of the home user.

The work which most closely addresses our project is manual troubleshooting guides that can be found on the Internet. Such troubleshooting guides are written by operating system providers [18], Internet Service Providers [10] or people willing to put their expert knowledge up for all to use [2, 14]. These documents describe steps for a user to manually follow to diagnose the source of network problems. They address many of the problems we chose to diagnose and were, in fact, our main source of motivation. Our goal in this project could be looked at as trying to automate many of the actions described in such troubleshooters.

10 Conclusion

Although Why is limited in the scope of problems it can diagnose, it does demonstrate the feasibility of developing a tool which encapsulates some of the expert knowledge of system administrators. While the tool cannot perform advanced network analysis, it can perform the basic steps users are asked to go through when they experience network difficulties. As such, we have developed a valuable tool that can simplify the process of diagnosing network problems for less knowledgeable users and for users who are knowledgeable about networks but do not use network tools often enough to have their usage memorized.

11 Acknowledgments

Many thanks to the Fall 2004 CSE561 Networks class for all their good questions. We would also like to thank David Wetherall and Valentin Razmov for the good feedback on our initial project plan. Finally, we would like to thank Andy Collins and Krishna Gummandi for their suggestions of tools and approaches we could use.

References

[1] T. Anderson, R. Mahajan, N. Spring, and D. Wetherall. *tulip: a tool for user-level internet path diagnosis.* `http://www.cs.washington.edu/research/networking/tulip/`.

[2] *Diagnosing Internet Connectivity Problems.* `http://www.losurs.org/docs/tips/general/connectivity`.

[3] Fyodor. *nmap(1) - Linux man page.* `http://www.die.net/doc/linux/man/man1/nmap.1.html`.

[4] Krishna Gummadi. Personal communication, 8 Nov 2004.

[5] D. Hagerty. *Net::Traceroute - traceroute(1) functionality in perl.* `http://search.cpan.org/~hag/Net-Traceroute-1.08/Traceroute.pm`.

[6] *host(1) - Linux man page.* `http://www.die.net/doc/linux/man/man1/host.1.html`.

[7] *netfilter/iptables project homepage.* `www.iptables.org`.

[8] V. Jacobsen. *traceroute(8) - Linux man page.* `http://www.die.net/doc/linux/man/man8/traceroute.8.html`.

[9] V. Jacobsen. *Pathchar.* `http://www.caida.org/tools/utilities/others/pathchar/`.

[10] *My Service is Down.* http://support.speakeasy.net/cgi-bin/support.cfg.

[11] *NIST Net Home Page.* http://snad.ncsl.nist.gov/itg/nistnet/.

[12] D. Oppenheimer, A. Ganapathi, and D.A. Patterson. *Why do Internet services fail, and what can be done about it*, in: *4th Usenix Symposium on Internet Technologies and Systems*, 2003.

[13] *ping(8) - Linux man page.* http://www.die.net/doc/linux/man/man8/ping.8.html.

[14] *Quick HOWTO - Simple Network Troubleshooting.* http://www.siliconvalleyccie.com/linux-hn/network-trouble.htm.

[15] *route(8) - Linux man page.* http://www.die.net/doc/linux/man/man8/route.8.html.

[16] M. Sabin, R.D. Russell, and E.C. Freuder. *Generating diagnostic tools for network fault management*, in: *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management V : integrated management in a virtual world: integrated management in a virtual world*, 1997.

[17] TCPDUMP public repository. http://www.tcpdump.org/.

[18] *Troubleshooting Internet Connection Sharing in Windows XP* http://support.microsoft.com/default.aspx?scid=kb;en-us;308006.

[19] F. van Kempen and B. Eckenfels. arp(8) - Linux man page. http://www.die.net/doc/linux/man/man8/arp.8.html.

[20] M. Welsch and A. Cox. netstat(8) - Linux man page. http://www.die.net/doc/linux/man/man8/netstat.8.html.