A TOOL FOR DIAGNOSING INTERNET CONNECTIVITY PROBLEMS

Masaharu Kobashi and Rosalia Tungaraza

University of Washington
Washington, WA 98105

Abstract

One of the major problems in the Internet today is that there is little support for identifying causes of problems experienced by users in end-to-end performance. To solve this problem, we designed and built a user-friendly system integrating a number of existing tools into a single application which is capable of diagnosing major problems that users are likely to encounter in daily operations of web browsers. As a result of our effort, we successfully built a working system that meets our goals, namely it should be easy to use and produce informative outputs outlining the diagnosis. At the same time we learned how to use well-known network tools such as tcpdump, traceroute, and others. We also discovered some unexpected realities of the Internet, which pose as obstacles for probing it efficiently. For instance, some nodes tend to filter out ICMP/UDP packets, while others have constantly revolving multiple aliases for a single publicized host name. Those are protective measures taken by host and network administrators, which prevented us from conducting proper measurement of some network characteristics.

1. Introduction

In recent years the Internet has become a part of our daily lives and one of the most important tools for communication and spreading information. However, on some occasions we either can't access the internet or can't retrieve data from web servers on the Internet. In such situations, there is almost no support for diagnosing what the main cause for such problems could be [4].

Although there are a number of useful tools dedicated for particular purposes, average users do not know what tools to use for what problems. On top of that, some of these tools are difficult to use while others produce outputs that are difficult to interpret. As if that wasn't enough, other tools require root privileges to operate. At the end of the day, most Internet users are left clueless about what might have disabled them from connecting to the Internet or from retrieving a particular file from a web server. Our study is motivated by the desire to alleviate these problems by creating a system that helps average users understand what causes their network connectivity problems.

In our attempt to build such a system we limit our scope in two ways. First, we focus only on web browsing, since it is probably one of the most frequently used Internet services. Secondly, we limit our target operating systems to Linux only. The latter constraint arose from the fact that we only had a limited amount of time to work on this project. From our perspective, Linux provides the greatest number of freely available tools for probing the network. Its operating system's source code and the code for those network tools are open, meaning that we could poke

around them especially when we entered unfamiliar regions in system building. Lastly, our system uses existing tools as components in order to avoid reinventing the wheel.

In terms of our goals, we wanted the system to be easy to use for users without special computer networks knowledge. Therefore, operationally the system needed to be simple and output-wise it was supposed to be easy to interpret. Second, we wished to gain practical knowledge and skills in using tools for network diagnosis and experience the reality of the Internet as active investigators of its characteristics. Towards the end of this project we evaluated our goals and discovered that we successfully attained them.

The rest of the paper is organized as follows: section two contains related works. Section three is a list of the possible causes of network connectivity problems. Section four is a summary of the diagnostic process followed by our tool. Section five is a detailed outline of how each step in the diagnosis is carried out and what software are used. Section sex is our evaluation of the functionalities and performance of the tool. Section seven is a description of a look-up table we tried to create for analyzing server availability. Section eight concludes our paper by summarizing what we learned and experienced through this project.

2. Related Works

To the best of our knowledge there is no published work in which somebody attempted to implement a tool similar to ours. However, there have been attempts to use similar tools to the ones we used to measure some of the network characteristics we measured.

For example, [1] also used ping and traceroute to investigate remote host availability and trace the path to it. They also had to tolerate the imperfections of these tools. Nonetheless, they didn't report the fact that some sites filter ICMP packets, while others give these types of packets less priority. Nonetheless in [3] [13] these problems with ICMP packets were reported.

While available tools are plenty for network measurements [2], there are very few tools which fit our objectives in measuring the network states such as bandwidth, bottlenecks, congestion. For example, a well-known tool, pathchar works in the single-end mode (i.e. works without cooperation of destination)[5] [6], it needs heavy overhead and is very slow. In our experiments, it took around 10 minutes to get a result for a path made up of 12 hops. Pipechar is another network diagnosing tool that works in the single-end mode [12]. However, it is also slow to be used in the real-time settings. Our ideal tool for these types of measurements was pathneck [6], which we used for locating bottlenecks on a path in the single-end mode within 10 seconds (in most cases).

3. Possible Causes of Network Connectivity Problems

Possible symptoms of network problems can be divided into two main classes: slowness and no response. The location of the causes of the problems is one or more of the following three areas: Local host, network, and remote host. Table 1 is an exhaustive list of possible locations. However, it is practically impossible to make an exhaustive list of concrete causes of the problems within each of the three areas.

Table 1: List of Possible Network Connectivity Problems

		No response	Slow
Problems in Local Host	Troubles in operating system (kernel)	X	X
	Troubles in the web browser in question	X	X
	Network interface is not configured or not existing	X	
	Local host's motherboard failure (some parts failed)	X	
	Incorrect routing table contents (e.g. wrong default gateway etc.)	X	
	Network service is not started (wrong initd scripts' contents etc.)	X	
	Local host's inappropriate TCP window size setting		X
	Non−existence or incorrect contents of resolver related configuration files	X	
DNS	DNS servers are not working	X	
	DNS servers are not reachable	X	
	Target remote host name is wrong (not in any DNS server's record)	X	
	Bugs in DNS lookup program such as dig, host, and nslookup used by browser	X	
	Remote host not working (for any reason, OS failure, hardware failure etc.)	X	
	Remote host's network service alone is not working	X	
	Remote host is overloaded		X
	Remote host's TCP window size setting inappropriate		X
	Remote host's HTTP server alone is not working	X	
	Path to and/or from the remote host is not reachable	X	
	Path to and/or from the remote host is congested		X
	Bandwidth of the path to and from the remote host		X
	Location of bottlenecks in the path to the remote host		X

(Note: Blue colored problems are diagnosed in our system, while black colored ones are not.)

In listing concrete causes, the best we can do is to enumerate very likely causes of slowness and no-response problems in each area. Table 1 shows the list of frequently occurring problems and their relations with the two symptoms mentioned above.

For the purpose of the categorization these problems in our system, we divide the problems into four classes, local host related problems, DNS related problems, remote host related problems, and network path related problems as shown in the table. It shows our system handles only a subset (blue colored ones) of those frequently occurring problems.

We wanted to include the TCP related parameter settings in our scope, since it is well known that they (e.g. the window sizes of TCP) affect the network performance significantly [6][10][11][14]. However, we curtailed that attempt due to time constraint and focused on the blue colored problems only.

4. Summary of the Diagnosis Process

Before delving into the bits and pieces of our tool, we present a flow chart that depicts how the system was set-up to analyze Internet connectivity problems. After detecting that there is no response from a given server, one would start with checking the local network settings. If these are configured correctly, move to checking whether there is any DNS related problem. If this in turn is not the problem, one would check whether the remote host is up and running. If it is, then the problem is most likely in the path characteristics between the user and the remote host. Thus, the tool provides a list of bottleneck paths along the route between these two nodes (see Fig. 1).

Similarly, if one can receive a web page, but the speed is much slower than what it normally is for the same page, then one can start diagnosing the problem by checking the availability of the remote host. Thereafter, one could check the path characteristics between the source and the remote host (again see Fig. 1).

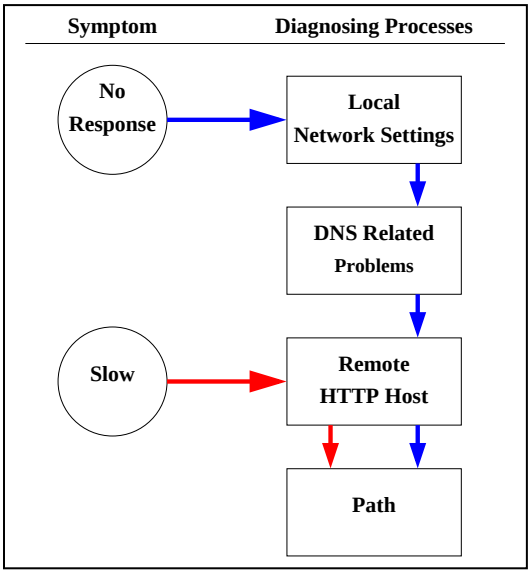

Figure 1: A flow chart outlining the steps taken in our tool to allow a user to diagnose why she/he can't retrieve a webpage from a specific web server or access the Internet in general

5. The Diagnosis Process

Now we will expound on the underlying design of our tool that enables users to walk through the flow chart presented above. We must stress that it is important to make sure the inaccessibility of a given web page is not caused by misspelling the web address or a particular wire necessary for connecting to the internet (depending on the user system's set-up) being unplugged or plugged into the system incorrectly.

5.1 Local Network Settings

In order to investigate the user's local network settings, our tool first checks whether the network interfaces were configured properly. It does this using **ifconfig** a command found in UNIX and UNIX-like operating systems e.g. LINUX. This command can be used to both assign an IP address to a network interface or look-up what the IP address for a particular network interface is (refer to LINUX man page for detailed description). In our tool, we use the latter functionality.

After checking the network interfaces' configurations and finding that they are configured correctly, the tool takes the user to the next step namely checking whether three main network related files contain essential information. These files are **host.conf**, **resolv.conf**, and **nsswitch.conf**.

The host.conf files resides in the /etc directory and harbors information that directs the resolver Our tool makes sure that the order specification line in host.conf includes at least **bind** so that it enables the resolver to use DNS.

Similar to the host.conf file, the resolv.conf file also resides in the /etc directory. Its main purpose is to provide the IP address of the name server that is to be used if the **bind** parameter is specified in the host.conf file. Thus, in this file, our tool makes sure that the IP address of the name server is correctly specified.

Lastly, the nsswitch.conf file, as the name indicates, is a configuration file for the name service switch. It directs the NIS (Network Information Service) to sources and the order in which to use those sources for looking up information about hosts, ipnodes, users and so forth. It too resides in the /etc directory. Our tool makes sure the "hosts" line includes at least "dns".

On the other hand, if the network related configuration files are correctly configured, our tool directs the user to check the routing table. The underlying command that enables this functionality is **route** (see UNIX man page). We use this command to both check what gateway the user's system uses for its packets and whether that gateway was correctly specified.

A summary of this process can be visualized in Figure 2. Notice that after every stage in checking the local network settings, if the outcome is negative, the tool prints out information stating that the check-up detected a problem and provides an explanation about the possible cause(s) of that problem. In situations were there is no problem with the local network settings, the tool directs the user to the next step of the diagnosis: checking whether there is any DNS related problem.

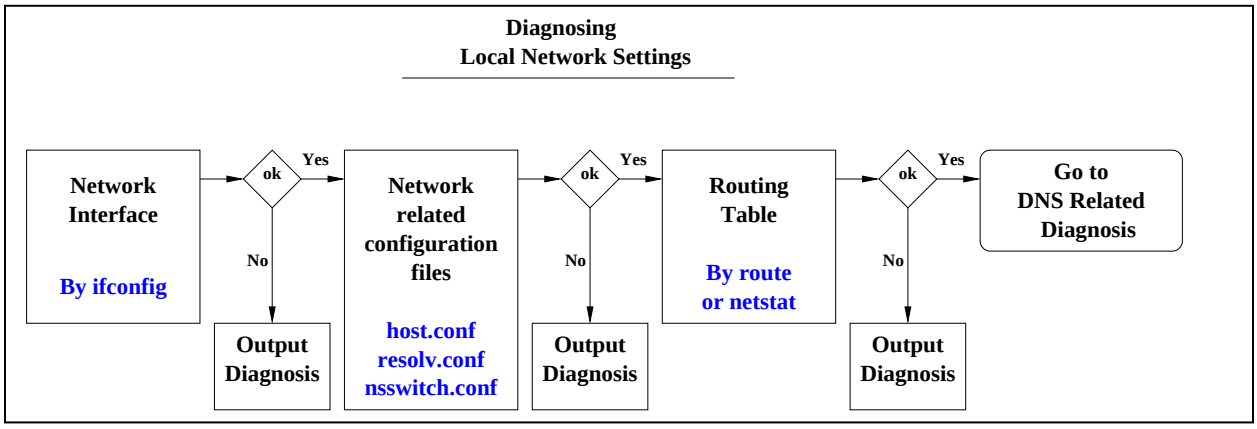

Figure 2: A flow chart outlining the steps taken when diagnosing the local network settings.

5.2 DNS Related Problems

The first step in diagnosing DNS related problems is to look up the IP address of the remote host within the DNS servers. The DNS servers to query are listed in one of the configuration files introduced above namely resolv.conf.

There are a number of DNS look up tools. The best known among them is **nslookup**. But it is now being deprecated. For example, in some recent Linux distributions such as Suse, whenever nslookup is invoked, the system outputs extra warning lines that say "nslookup is deprecated ... Consider using the 'dig' or 'host' programs instead."

We chose **dig** for two reasons. First, its output is designed to be parsed in a consistent manner. Second, unlike the other two tools, it also reports all intermediate DNS servers that are recursively called while resolving a given IP address.

Figure 3 depicts how we employed dig in our tool and how we go about diagnose DNS related problems. Shortly, the address of the target remote host is fed to dig. If it successfully finds the IP address corresponding to the name of the remote host, the investigation at this stage is over. If it fails, then the system checks if the last referred DNS server is reachable by invoking traceroute. It then reports the result on a diagnosis window for the DNS category.

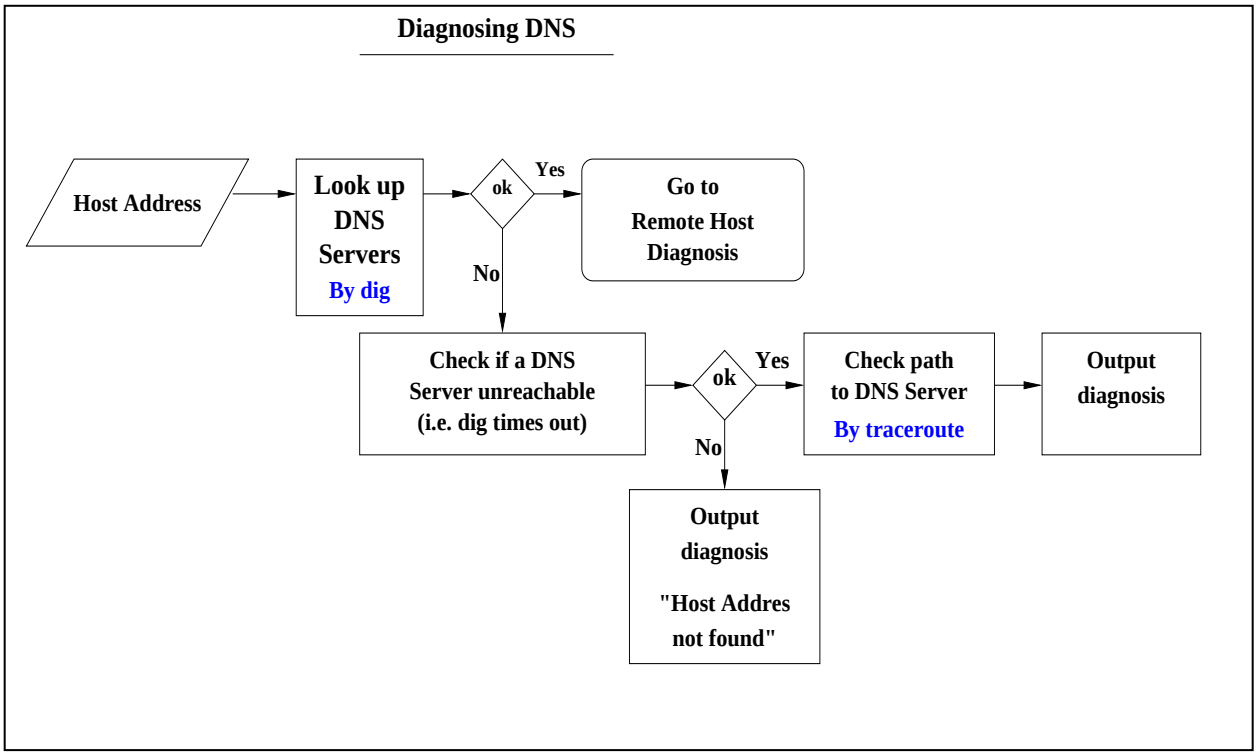

Figure 3: Diagnosing DNS related problems

5.3 Remote Host Availability

Our system does two operations in investigating the state/availability of the remote http host (see Figure 4). One is to check whether it has an active connection to the network. This is done by **traceroute**. If traceroute fails to get a response from the remote host, the system reports that problem.

In such a case, the tool doesn't just report: "Remote host not functioning" because failure to get a response from the remote host can be caused by many factors other than a remote host being down (not functioning). Other possibilities include a problem in some links/nodes in the forward path between the last successful hop and the remote host, or one in some links/nodes in the backward path from the remote host to the local host. It might also be caused by the policy of some nodes that is functioning normally but is set not to respond to such packets with TTL created by traceroutes or ping. Therefore, the system reports all of the above possibilities in its diagnosis.

If traceroute reports successful connection with the remote host, our system performs the second operation: evaluate the network delay and the remote host delay. To do so, we compare the RTT of ping with RTT of an http request. If the RTT of http request is substantially greater (e.g. more than 100 ms) than the RTT of ping, it can be inferred that the remote host is overloaded. But as we will explain later, this assumption doesn't always hold and might even be invalid at other times.

The way our system calculates the two mentioned RTT is by first starting **tcpdump**. Tcpdump inspects each packet passing the kernel. While tcpdump is capturing packets that arrive at the kernel, our tool sends out ping packets and http packets to the remote host. The latter are being sent by a http client module in our system. The response from the remote host is picked up

by tcpdump, which directs those packets to our tool. Our tool then calculates RTT for each transaction.

The http client module we used was created from http library bundled in tcl language. It allows us to send any http message in the http protocol. For this measurement we tried both HEAD and POST. HEAD tends to be cached in the remote host and prevents us from measuring the real state of overload in it. But the response to POST cannot be cached. It requires the remote host to do some decision making depending on the contents of POST request. Our system sends a meaningless garbage text. It always results in a reply with an error message. Therefore, it is likely that the responds to POST reflect the state of overload more truly than the responds to HEAD request.

The result of the measurements is reported in the diagnosis window of the remote host category (second diagnosis window from the bottom in Figure 4).

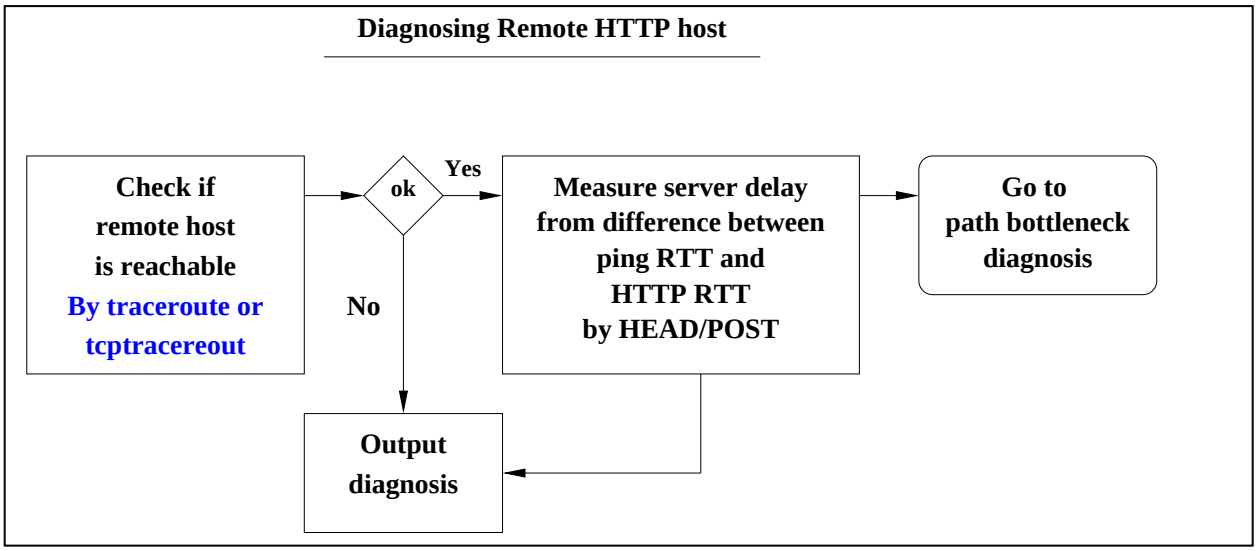

Figure 4: Diagnosing remote host related problems

5.4 Path Characteristics

Investigating path problems is a very broad and sophisticated area in network connectivity problem diagnosis. Therefore, we limited our attempts in this section to tracing the path to the remote host and locating bottlenecks along that path. For the former we used **traceroute**, and the latter we used a tool called **pathneck** which was developed by a group at Carnegie Mellon University.

An interesting discovery that we made with traceroute was that even though one may not be able to receive any response from some intermediate node along the path, traceroute may still report the path to the destination host as reachable (See Figure 5 for such an example).

```
1  eureka-GE1-5.cac.washington.edu (128.208.4.100)  1.225 ms  0.203 ms  0.124 ms
 2  uwbr1-ge2-2.cac.washington.edu (140.142.155.23)  1.053 ms  0.468 ms  0.362 ms
 3  prs1-wes-ge-1-0-0-0.pnw-gigapop.net (198.107.151.30)  0.989 ms  0.600 ms
0.479 ms
 4  att-pwave-1.peer.pnw-gigapop.net (198.32.170.29)  0.496 ms  0.557 ms  0.483
ms
 5  tbr2-p012502.st6wa.ip.att.net (12.123.203.178)  1.628 ms  3.188 ms  0.985 ms
 6  tbr2-cl1.cgcil.ip.att.net (12.122.10.61)  43.238 ms  41.548 ms  41.693 ms
 7  tbr1-cl2.cgcil.ip.att.net (12.122.9.133)  42.096 ms  42.057 ms  42.454 ms
 8  tbr1-cl1.n54ny.ip.att.net (12.122.10.1)  61.442 ms  61.915 ms  61.195 ms
 9  tbr1-cl8.phlpa.ip.att.net (12.122.2.18)  64.566 ms  62.917 ms  64.696 ms
10  gbr2-p10.phlpa.ip.att.net (12.122.12.102)  64.440 ms  64.303 ms  64.307 ms
11  gar1-p370.phlpa.ip.att.net (12.123.137.25)  64.324 ms  64.284 ms  64.315 ms
12  12.118.191.18 (12.118.191.18)  67.070 ms  67.032 ms  67.063 ms
13  * * *
14  www.pair.com (66.39.3.7)  71.348 ms  71.283 ms  71.187 ms
```

Figure 5: Sample output from traceroute showing

From Figure 5 a single * mark means there was no response in 5 seconds. Hop number 13 has three consecutive * marks, which means three attempts to get a response using a UDP packet with TTL 13 failed. Some likely explanation is that hop number 13 did not respond or some link/node in the backward path from it either had some problem or is filtering ping packets. Nevertheless the destination, which was node number 14, was reachable and the local host was receiving its responses.

This means there can be nodes which are functioning correctly in delivering TCP packets but are filtering UDP probing packets or are set not to respond to TTL error. Our system is designed to cope with these discrepancies by not giving up at consecutive failures in the intermediate nodes while processing traceroute output.

The second goal in investigating path problems was explored using another ready-made tool called pathneck. There are more than a dozen tools freely available for measuring various network path characteristics. Well known tools for this purpose are **pathchar, pipechar, pathrate, pchar,** and **nettimer**. Most of them however, need at least five minutes and/or access to the the destination host.

Our system was designed for real-time diagnosis without needing access to the target remote host. Pathneck fulfills our requirements: can perform path analysis in less than 10 seconds (in most cases) and does not need access to the destination host. It uses what they call recursive packet train (RPT) to locate bottlenecks. Precise definition of "bottlenecks" and how they are located are explained in [6]. The raw output from pathneck is interpreted by our system and the result of the interpretation is displayed in the diagnosis window for the path category (See Figure 8).

6. Functionalities and Performance of the System

We have successfully incorporated seven independent network tools in a single application with a graphic user interface for ease of use as shown in Figure 8. This application also meets our goals:

1. It is easy to use by any one who does not have special skills

2. It presents the results of each diagnosis in a form which is informative and easily understandable to users without requiring any special knowledge

In addition to that, it also provides the raw output from each component tool for those who are interested in more details (See the rightmost window in the system).

One of the most interesting measurements by our system is the measurement of the remote servers delay by computing the difference between the RTT of ping and RTT of http requests. In our experiments most http servers show longer RTT for http request than for ping packets. This is logical because the operating system's scheduling mechanism makes ping responses faster. But with some http hosts, occasionally RTT of ping is longer than RTT of http request (HEAD or POST as explained in the previous section).

Such discrepancies can be explained by the following factors.

1. ICMP ECHO REQUEST packets and/or reply packets to them are given lesser priorities than TCP packets in some hosts and/or some nodes in the paths in question.
2. Target remote host has constantly revolving multiple aliases of its publicized host name. In such cases if the remote host is accessed by the publicized name, actual host accessed can be different depending on the timing of access. A sample server which showed this discrepancy very frequently was www.yahoo.com. That is why we chose it for the demonstration in our presentation.

The second factor can be eliminated by accessing the remote host by IP address instead of its host name. But our system is not configured to memorize the IP addresses of the hosts and use the same IP addresses in both ping and http transactions. We must elaborate here that we did not use the raw output from the ping program. The RTT given by ping is always a few milliseconds longer in our platform than the RTT measured at the stage where tcpdump captures packets. Our system computes RTT of ping and http transactions using the same timestamps displayed by tcpdump.

Despite the possibility for such discrepancies, good characteristic of our system is if it says "The remote host is overloaded", it is very certain that the remote host actually is so. It is because the abnormality causes only false negative of the diagnosis of remote host's overload. The likelihood of false positive is reduced by the factors described above.

Overall, despite the fact that our system was built for a course project with strict time constraints, it can be used as a foundation for developing a more sophisticated diagnostic tool. As for the bottleneck detection, the developers of pathneck claim that it is capable of locating the bottlenecks for 70% - 95% of paths from most of their probing sources. The details of their design and claims are in [6].

7. The Look-up Table

Towards the end of our project, we experimented with the idea of a look-up table that would add some level of convenience to the user when he/she is using our tool. The table was to enable the user to infer a server’s present availability based on that server’s past availability trend. It was also supposed to be very region-specific.

Data pertaining to specific web servers found in a defined geographical location were to be collected over a period of time. That data should include the average round trip time (RTT) taken by TCP packets from a specific host node to a specific web server. It should also indicate the relative availability of the server given a specific time period of a specific day of the week (see Figure 7 for a sample section through the look-up table).

We envisioned its use to be as follows: a user tries to access the web page www.mtholyoke.edu on a Saturday evening, but does not succeed. Then he/she can open the look-up table and see what that web server's availability trend is. If it has a very low availability on Saturday evening, but a high availability on Saturday night, the user may decide to wait till later and try accessing the web server again. However, if the server has a high availability trend on Saturday evening, the user may perform further tests using our tool to find out why on that particular Saturday evening he/she cannot access www.mtholyoke.edu.

In order to find out whether such a table is even feasible and what its implementation would entail, we limited the geographical regions we covered and the amount of IP addresses we used. We picked IP addresses ranging from 138.110.0.0 to 138.110.255.255, which were the complete set of IP addresses assigned to Mount Holyoke College (MHC) in Massachusetts. We also picked IP addresses ranging from 128.208.0.0 to 128.208.255.255, 128.95.0.0 to 128.95.255.255, and 148.142.0.0 to 148.142.255.255 from the University of Washington (UW) in Washington.

These were then pruned based on the criteria outlined below until we only had 149 IP addresses from MHC and 556 IP addresses from UW belonging to machines that harbor web servers actively listening for http requests (see Figure 6 for a summary of this process).

After having obtained all the 65025 IP addresses from MHC and 195075 IP addresses from UW, we used the command **host** to find out how many among those had been assigned a domain name. Host basically resolves an IP addresses and prints out its name as it appears in the DNS. This process left us with 12675 IP addresses from MHC and 70094 IP addresses from UW.

In order to minimize the amount of temporary IP addresses, we were extracting, we performed the above part of the experiment around 1 am on a Wednesday night. This ensured that most libraries (except the undergraduate library at UW) and other areas where one could get wireless Internet access where closed.

Then we randomly selected only 10305 IP addresses from the 70094 ones for UW. This was purely for manageability purposes given that we only used one computer to conduct this experiment. At this stage none of the MHC IPs were pruned. After that we used a perl module called **Network::Discovery::Scan** to scan all 12675 IP addresses for MHC and 10305 IP addresses for UW to find out which among those have servers on port 80 that were actively listening for requests. We ended up with 149 IP addresses for MHC and 556 IP addresses for UW.

These were then used to randomly send TCP probes using **tcptraceroute** from a computer stationed in Commodore Dutchess (within the UW campus) to all those servers during specific time periods of specific days of the week. From the tcptraceroute output we extracted the RTT for each probe (if successful) and the number of times each trial was successful for each server at MHC and UW. The RTT was afterwards averaged and a sample output can be seen in Figure 7.

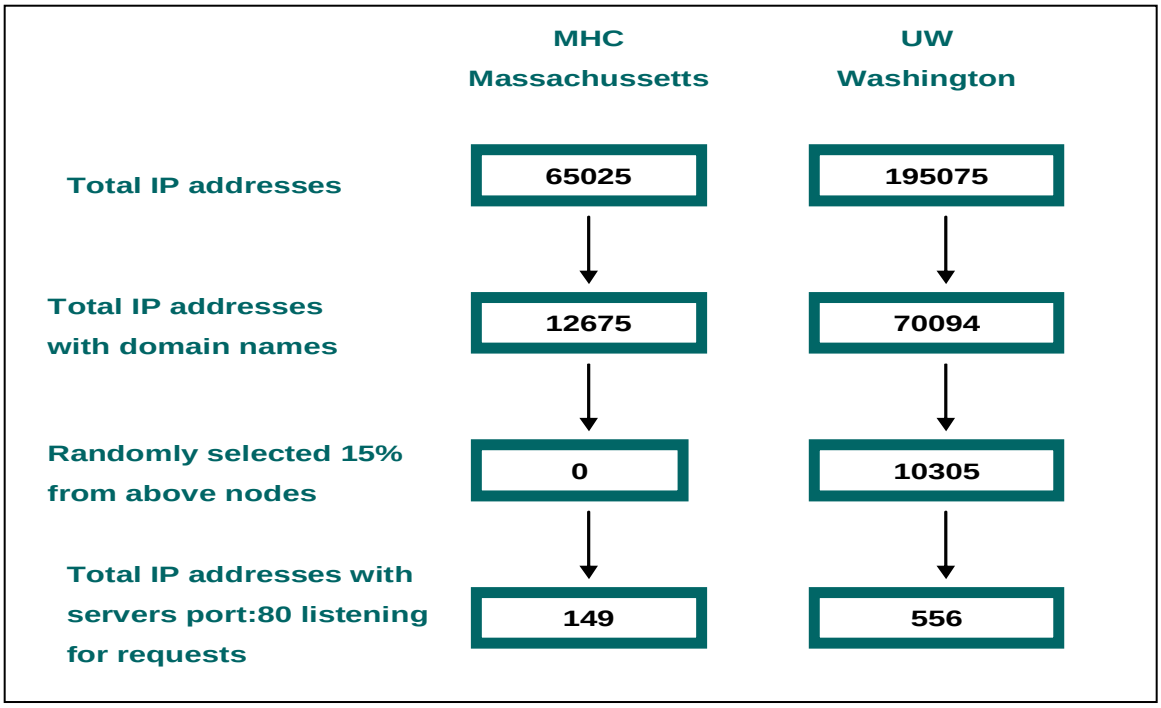

Figure 6: An outline of how we pruned the number of IP addresses used in the experiment to 149 for MHC and 556 for UW.

SATURDAY

	12:22am			6:00am			10:00am			12:30pm			4:40pm			7:40pm		
	AVG RTT (ms)	STDEV	SUCCESS	AVG RTT (ms)	STDEV	SUCCESS	AVG RTT (ms)	STDEV	SUCCESS	AVG RTT (ms)	STDEV	SUCCESS	AVG RTT (ms)	STDEV	SUCCESS	AVG RTT (ms)	STDEV	SUCCESS
www.mtholyoke.edu	76.8	1.76	6	76.1	0.89	6	74.65	3.79	6	76.5	3.22	6	75.5	1.03	6	76.9	2.452	6
www.academic.org	73.6	2.4	6	73.1	0.82	6	76.67	1.63	6	77.39	1.39	6	77.9	2.5	6	72.8	1.642	6
wins.mtholyoke.edu	75	3.48	6	76.9	3.16	6	76.94	3.05	6	80.76	7.98	6	76.2	3.6	6	76.7	3	6
webtest.mtholyoke.edu	73.5	2.9	6	72.9	1.23	6	76.12	0.89	6	74.31	4.11	6	77	1.81	6	74.5	4.068	6
snow.mtholyoke.edu	72	1.79	6	76.9	2.67	6	75.03	3.49	6	78.25	1.52	6	77.2	5.2	6	75.8	3.268	6
poe.mtholyoke.edu	77.6	2.25	6	78.2	1.6	6	77.49	2	6	77.65	1.86	6	77.7	1.83	6	77.2	2.366	6
pilot.mtholyoke.edu	75.1	2.27	6	75	2.12	5	79.08	4.56	6	75.92	3.79	5	78.9	3.31	6	77.3	3.363	5
kold.mtholyoke.edu	74.6	2.28	2	76.7	0.74	2	77.01	0	1	80.47	0.75	2	80.3	3.09	2	79.9	10.92	2
kn217-hp.mtholyoke.edu	78.1	3.47	6	74	2.29	6	75.71	2.82	6	78.43	3.12	3	76.7	3.58	3	75.3	2.701	6
kedance-hp.mtholyoke.edu	76.9	3.11	6	77.6	3.14	6	78.1	2.5	3	76.68	2.39	6	77.2	4.15	3	77.5	3.312	3
ke107hp.mtholyoke.edu	77.9	2.55	6	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Figure 7: A section through the Look-up table showing data collected and analysed for some web-servers at MHC on a Saturday.

Since we kept getting negative values for RTT of almost all probes sent to machines at UW, these data will not be included in our paper. We did find out that the negative values represented a RTT less than 1msec. Tcptraceroute reports its data in msec only.

In general, this part of our project serves only as an indicator that it is possible to create a look-up table that could be used for users to infer present server availability from their past availability trend. We don't encourage anyone to use our data as absolute values given that we conducted this experiment within two days only.

However, if anyone were to build such a table then he/she should consider changing the following to their methodologies. First, use more than one computer located in the same place for sending test probes to the servers. This will speed up the process tremendously and will facilitate the use of all IP addresses that have been assigned domain names.

8. Conclusion

There is a lot that we learned and experience through this project, most of which is summarized below.

8.1 Tools

During the early stage of the project, we discovered many network probing tools. Among those we only used seven for our system. In the process of evaluating the tools we found there are very few tools that could perform path diagnosis in real-time and in a single-end mode, which was one of the requirements for our system. Most path diagnosing tools need a few minutes to tens of minutes to investigate path problems and/or they require access to the destination host (two-end mode). Our choice, pathneck, was one of rare exceptions, although its capability has limitations as stated in the previous sections.

8.2 Reality of the Internet (unexpected obstacles)

We also discovered that a lot of network administrators and host administrators do take measures in protecting their systems from unnecessary work load caused by probes coming from external parties such as ourselves. These would often filter ICMP packets, especially ECHO REQUEST packets. Moreover, they give ICMP packets less priority in the waiting queues. Some even filter UDP packets from traceroute. www.amazon.com for instance, filters both ping and traceroute packets.

These administrative policies make the data collected by ping or traceroute (e.g. RTT) unreliable or unusable. In fact, we attribute the difference in RTT between packets send by ping and http requests to this phenomenon (recall section 5.3).

Another unexpected finding we made is that some site administrators tend to use multiple aliases for one host name. In other words, they use them interchangeably, each within a given time period. For example, both www.yahoo.com and www.cnn.com have eight aliases for their single host address as of this writing. The ordering of the aliases is constantly reshuffled. This behavior can be another cause of the illogical differences between the RTT of ping and RTT of http request when the target remote host is specified by host name instead of raw IP address.

8.3 Difficulties of the system building

Unlike usual applications, our system needs to investigate data owned by the kernel. Some tools in our system, such as tcpdump, require root privilege to execute, which a general system user doesn't have. Thus, in order to make our system usable by the general user, we needed to circumvent that barrier.

One way was to use **suid**. But this was not enough, since it applied only to the particular executable program that was set to suid. Any library function, which required root privileges, could not be called from that executable. To solve this problem, we used a tool called **sudo** which was capable of fooling the operating system by making any user a "root user" when the user is executing a particular program.

Another difficulty which we experienced was how to efficiently integrate the component tools we chose for our system. Each of these tools had its own I/O behavior in terms of buffering and timing. Some such as traceroute had unexpected output behavior. They combined standard output with standard error in an unusual way. We finally solved this problem and in the process increased our knowledge and skills of software engineering.

Overall we think that this project was very enlightening and a great educational experience.

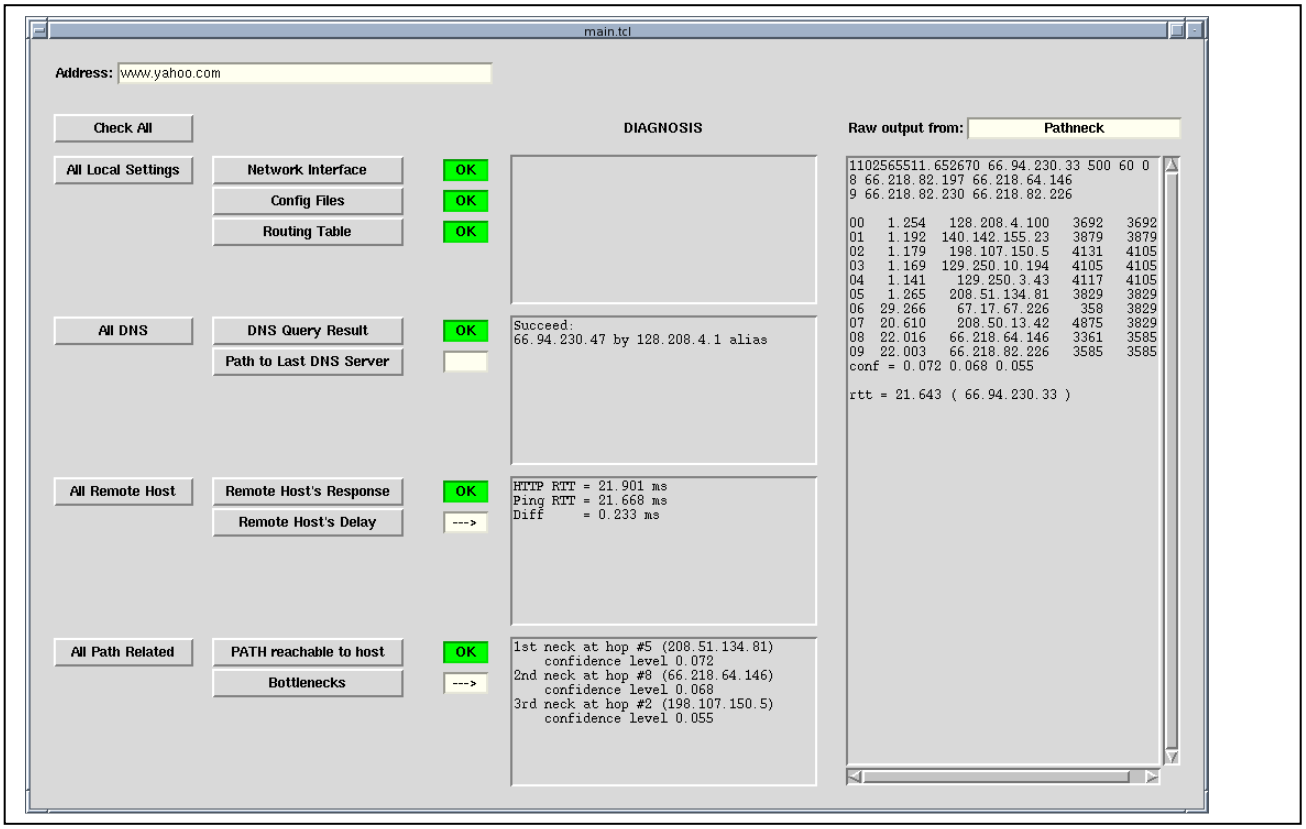

Figure 8: The GUI of our system

References

[1] K. Anagnostakis, M. Greenwald and R. Ryger. “cing: Measuring Network-Internal Delays using only Existing Infrastructure”. In Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003).
[2] O. Bonaventure. “Software Tools for Networking”. IEEE Network Jan/FEb 2002.
[3] T. Chen, and L. Hu. “Internet Performance Monitoring”. In Proceedings of the IEEE, August 2002.
[4] D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. “A knowledge Plane for the Internet”. SIGCOMM 2003.
[5] A. Downey. “Using pathchar to estimate Internet link characteristics”.
[6] N. Hu, E. Li, Z. Mao, et al.”Locating Internet Bottlenecks: Algorithms, Measurements, and Implications”.
[7] K. Lai and M. Baker. “A Tool for Measuring Bottleneck Link Bandwidth”.
[8] K. Lai and M. Baker. “Measuring Bandwidth”. IEEE, 1999.
[9] R. Mahajan, N. Spring, D. Weatherall and T. Anderson. “User-level Internet Path Diagnosis”. SOSP, 2003.
[10] J. Nolot. “End-to-End Packet Delay and Loss Behavior in the Internet”. SIGCOMM. pp. 289-297, 1993.
[11] J. Padhye, and S. Floyd. “Identifying the TCP Behavior of Web Servers”. In Proceedings SIGCOMM '01, June 2001
[12] V. Ribeiro, R. Riedi, and R. Baraniuk. “Locating Available Bandwidth Bottlenecks”.
[13] S. Savage. “Sting: A TCP-based network measurement tool” Proceesings of the 1999 USENIX Sumposium on Internet Technologies and Systems, pages 71 - 79, Oct. 1999.
[14] S. Shakkotai, R. Srikant, N. Brownlee, A. Broido, and K. Claffy. “The RTT Distribution of TCP Flows in the Internet and its Impact on TCP-based Flow Control”. http://www.caida.org/outreach/papers/2004/tr-2004-02/tr-2004-02.pdf, 2004.