CS 413 - Advanced Networking and Telecommunications

Lab 02: Traceroute

Anna Millerhagen, Adam Stammer

(Dated: February 26, 2020)

Abstract

We were first tasked with finding the ip route, and maximum transfer unit, from our lab workstation to another workstation in a different room. We didn't know how far away it was or what the existing network layout looked like, and we were limited to using only the 'ping' utility command. Once this route (of 14 hops), and mtu (of 1472 bytes) was discovered, one ping command at a time, we moved on to further experimentation of network discovery using the 'tracert' utility command. Seeing similar results on our local network, we then moved on to global networks even tracing to and from other countries. We then repeated these experiments with IPv6. Through all of this we gained a better understanding of network architecture as a whole, and how wide area networks communicate.

I. PING

A. Configuration and Definitions

To begin this lab we configured our computer systems to use DHCP to connect to the network. IPv4 and IPv6 were both active throughout this lab.

All references to a "node" or "system", in this lab, refer to Layer 3 or above machines. That is to say that they are machines concerned with packets, and do have IP addresses of their own.

B. Ping - Maximum Transmission Unit

Once connected to the network we had two tasks. The first was to find the Maximum Transmission Unit (MTU) between our system and a specific workstation in another room. To do this we used the -f command flag to indicate that the packet sent should not be fragmented. We also used the -l [xxx] command flag to force the packet be of the indicated size.

In the case of the packet being too large and requiring fragmentation to continue, the ping fails, and we got a response indication as such. This tells us that the packet is larger than the MTU. An example of this kid of output can be see below.

```
C:\Users\Adam> ping 192.168.1.1 -f -l 2000
Pinging 192.168.1.1 with 2000 bytes of data:
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

On the other hand, if the packet was small enough to reach the destination without fragmentation, it just appears as a normal ping response. This indicates that the packet

size was smaller than or equal to the MTU. We knew we were searching for the lower side of this fence. Guessing at a starting range, we used a process similar to bisection to narrow down the proper packet size to just one byte less than the smallest packet size that requires fragmentation. In our case, we found that 1472 bytes was the largest packet we could send without a fragmentation error occurring, thus the MTU between the systems in question was 1472 bytes. We noticed that this process would be relatively simple to automate in a script.

C. Ping - Path Tracing

The next task was to find a network path from our system to the same remote system used in Part 1. We spent a considerable amount of time reading the command descriptions and manual pages hoping for something to stand out. After testing multiple subcommands built into the ping command, each one failing, we began to think it may require more than one command to accomplish. With this in mind, we eventually hypothesized taking advantage of the Time To Live (TTL) of each packet. Generally speaking, each hop from one node to the next decrease the packets TTL by one, and when the TTL reaches zero the packet dies. So, if we send a packet with a TTL of one it should die at the node adjacent to our system (in this case the router). A TTL of 2 should kill it at the next adjacent node and so on, each one reporting back its IP address and a message indicating a packet's TTL expiration. An example of this response can be seen below

```
C:\Users\Adam> ping -i 1 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 10.82.17.1: TTL expired in transit.
Reply from 10.82.17.1: TTL expired in transit.
Reply from 10.82.17.1: TTL expired in transit.
Reply from 10.82.17.1: TTL expired in transit.

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

Knowing this, all we had to do was increase the TTL by one, each time getting closer to reaching our goal. Eventually the ping would succeed once we reached the remote system. This requires an additional ping command for each node in the path, but revealed the order and IP addresses of each node. Like before, we believed this process would be simple enough to automate with a script.

Not all of the nodes in question reported their IP address back, but after 14 hops we did eventually reach the target system. You can see this path below. We believe this is a node specific configuration, as some other target systems would indeed reveal the entire path (8.8.8.8 for instance).

IPv4 Address	TTL	IPv4 Address	TTL
199.17.162.1	1	—	8
199.17.175.10	2	—	9
199.17.175.43	3	—	10
—	4	—	11
—	5	—	12
—	6	—	13
—	7	192.145.254.134	14

II. TRACEROUTE

'Tracecert' is a windows command that accomplishes the path finding done above in a single command. It uses the same method of varying TTLs as we used. Below you can see the hop counts of various targets.

Target	Hop Distance
www.bemidjistate.edu	7
192.145.254.134	14
199.17.161.32	2
www.uio.no	19
www.kame.net	16
203.178.141.194	16
www.traceroute.org	16

There weren't really any surprises in these results but we did run this command multiple times on some of the targets and what we found was interesting. If you run the command twice quickly, the path seemed to be the same, but if you wait a while between commands, then the paths were often different, at least for a part. This was more noticeable in the traceroutes of the foreign targets (uio.no and kame.net). We believe this is due to the arp caches of various nodes in the path going stale after some time, and even some nodes going offline or comming online, leading to a new route being found on successive traces.

A. Reverse Traceroute

Our next task was to try and find the route from somewhere else back to us. To do this we used www.traceroute.org which lists multiple route sources from various locations. We supplied them with our own IP address and they ran similar scripts to find that reverse route. We began with IPv4 and found few routes to be symmetrical. That is to say that the route from our system to theirs, was different than their route to us. These paths were also frequently lopsided such that it took fewer hops to get from us to them than back. We weren't surprised by the different paths but we still don't really have a good guess as to why our paths there are so frequently shorter. We also found paths tended to be more different the longer the path was, like those that go overseas. This was as expected but nice to see in practice nonetheless.

From there we moved onto IPv6. This proved more difficult than anticipated. Many of the IPv6 target systems were unreachable for one reason or another. Sometimes we could reach them from our system but a reverse traceroute was unsuccessful. It's hard for us to know what caused all of these issues but our best guess is that the lack of IPv6 nodes between our system and those in question may be more prone to congestion. Pretend an IPv6 node is the only IPv6 connection between the Minnstate network and HBC. It would make sense to us that that node could become a bottleneck for IPv6 traffic, but this really is just a guess. Of the few IPv6 traces that did complete we found them to frequently be slightly shorter than similar IPv4 routes, and that the IPv6 routes tended to be more symmetrical. Again we guess this is due to a fewer number of IPv6 nodes to route through. This was less of a trend with oversea targets, again possibly due to more possible nodes in a longer path.