Something

IN3260

Oskar Haukebøe

December 8, 2023

1 Introduction

Network problems can often be obscure and challenging to diagnose for end users. When experiencing problems, it can be difficult to know where the problems emerged. The bottleneck could be at the local WiFi, or it could be that one is just accessing a distant server, and therefore is experiencing slowdowns.

One project which aims in helping to detect whether the bottleneck is on the local WiFi, or somewhere else is NETHINT [1]. This project aims to passively listen to all WiFi traffic and to be able to compare the latencies and packet losses for all devices on the WiFi network, as described in [2]. By doing this, it should be able to help detect where the bottleneck is.

In this assignment, the aim is to verify whether the data collected by NETHINT can determine whether the bottleneck is on the local WiFi. This is done by using a TEACUP [3] testbed to perform automated experiments on actual machines with different network configurations.

2 IN-PROGRESS The testbed

In order to test whether the NETHINT program can determine where the bottleneck is situated, it is necessary to have it listen to various traffic with the bottleneck at different places. For this reason, we have set up the testbed as shown in Figure 1. This represents a situation where there is one user Client 2 who is streaming a video from Server 2, and another user Client 1 who is doing something else that is consuming bandwidth from Server 1. The goal is to be able to determine whether the traffic between Server 1 and Client 1 is affecting the traffic between *Server 2* and Client 2.

All nodes in Figure fig:topology are physical computers running Debian Linux, except for the Switch, and are being controlled by another computer not in the figure, using TEACUP. This makes it possible to automate the generation of network traffic, the throughput, and delay at the routers, as well as logging the network traffic.

We are using a switch between the two clients and the first router in order to have the clients both be accessed through the same network interface at Router 1 when running over the wired connection. This allows us to easily limit the bandwidth to both clients at the router and have them share the bandwidth.

The WiFi AP is also connected to the switch and hosts an unencrypted WiFi network. The network is unencrypted because the NETHINT program does not support decrypting the network the WiFi traffic. The two clients are connected to this WiFi

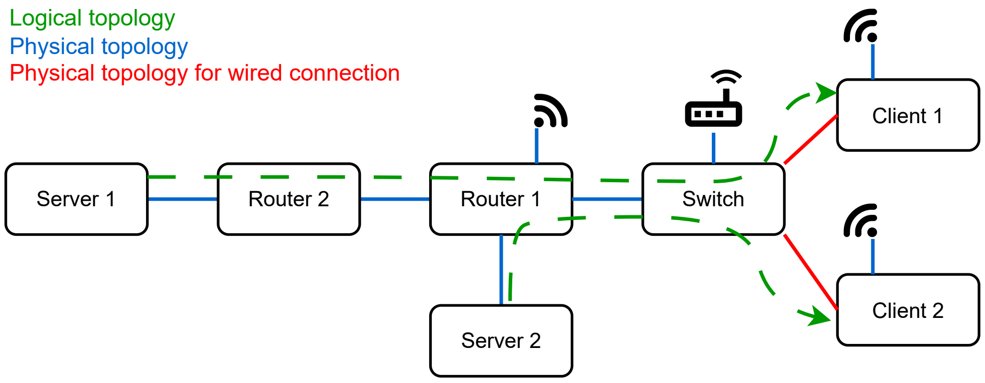

Figure 1: Topology of the testbed

network and they can reach the two Servers over it. The WiFi card at Router 1 is set to monitor mode and is used by NETHINT to listen to all the WiFi traffic between the clients and the WiFi AP. In order for the router to listen to the WiFi traffic, the WiFi interface must be set to use the same channel/frequency as the router. Both the WiFi interface at Router 1 and the WiFi AP have therefore been set to use channel 3.

In order to tell the machines how to reach each other, they had to be configured to know where the packets should be sent. As an example Client 2 got configured as follows in the file `/etc/network/interfaces.d/vlan11-iface`:

```
auto enp36s0
iface enp36s0 inet static
      address 172.16.12.5/24
      up route add -net 172.16.10.0 netmask 255.255.255.0 gw
         ↪ 172.16.12.254 || true
      up route add -net 172.16.11.0 netmask 255.255.255.0 gw
         ↪ 172.16.12.254 metric 10 || true
      up route add -net 10.10.12.0 netmask 255.255.255.0 gw
         ↪ 172.16.12.254 metric 10 || true
```

where `172.16.12.254` is the IP address of the network interface of Router 1 facing the switch. `172.16.11.0` is the IP address of Server 2, and `10.10.12.0` is the IP interface of Server 1. This essentially tells the client that it can reach the two servers through Router 1. The address `172.16.10.0`, which is the interface on Router 1 facing Router 2, had to be added for TEACUP to run as it first wants some of the computers/interfaces to ping each other, and so it had to be possible for Client 2 to ping this interface on Router 1.

The configuration above sets up the routes for running the tests over the wired connection. But we also wanted to run the tests over a wireless connection, which is why `metric 10` is also specified in the configuration above.

As the two clients now can reach the servers through both the WiFi AP and through the cables, they need to be configured to use the correct network interface. This makes it possible to run the command

```
sudo ip route add 172.16.11.0/24 via 172.16.13.1 metric 2
```

on the client in order to send the traffic over the router, which has IP address `172.16.13.1`, instead of over the cable. Similarly, one can then run

```
sudo ip route del 172.16.11.0/24 via 172.16.13.1 metric 2
```

in order to use the wired connection again.

2.1 The bottlenecks

The bottlenecks are placed at either Router 1, Router 2, or at the WiFi AP. This corresponds to the two users at Client 1 and Client 2 either having a common bottleneck or not having a common bottleneck. When the bottleneck is at Router 2, the user at Client 2 should not be affected too much, as Router 1 should still have room for more bandwidth than is being used.

In order to configure the bottlenecks, we change the throughput at Router 1 and Router 2. The throughput at the WiFi AP is set to the lowest value it supports which is 54 Mbps. The values at Router 1 and Router 2 are then set either higher than this or lower, depending on where the bottleneck should be. The values used at the routers for the three different bottleneck configurations are listed in Table 1.

Table 1: Throughput at the two routers in the different bottleneck configurations

Bottleneck at:	Router 1 (Mbps)	Router 2 (Mbps)
Router 1	15	70
Router 2	70	15
WiFi AP	70	70

While the throughput is the main way in which we decide where the bottleneck is, we also change some other variables. These are the queue length at Router 1 and the delay at Router 1. The queue lengths used are:

- 0.5 BDP
- 1 BDP
- 1.5 BDP
- 2 BDP

The tests were also run with both a delay of 50ms and 10ms at Router 1. Additionally, Router 2 always had a delay of 10ms.

In order to set these limitations, we used the built-in functions of TEACUP which allows us to set these limitations for the router, and then it runs the tests for each of the values specified. This is done by setting the following variables in the TEACUP configuration:

```
# Emulated delays in ms
TPCONF_delays = [25,]

# Emulated bandwidths (downstream, upstream)
TPCONF_bandwidths = [
    ('70mbit', '70mbit'),
]

# Buffer size
TPCONF_buffer_sizes = [31, 62, 93, 124]
```

This will run 4 different tests, as there are 4 different buffer sizes specified. Also, note that the delay here is set to 25ms. This is because TEACUP sets the delay for both downstream and upstream. Hence, the total delay for the rtt of the packets is 50.

When running the tests TEACUP sets up a mix between these lists and runs the tests. But this is not what we wanted to do, which is why only the buffer sizes have multiple values specified. As we wanted to use a different set of buffer sizes for each delay configuration, it would cause us to run more tests than necessary.

Additionally, TEACUP does not support setting different configurations for multiple routers. This meant that we had to limit the bandwidth for Router 2 in a different way. It was, therefore, easier to have multiple configuration files with different configurations, and then have a script running them.

In order to set the delay for Router 2, we used the `tc` program to set qdisc rules. TEACUP has a variable `TPCONF_host_init_custom_cmds` which runs commands specified on a specified machine. we used the following configuration for automatically setting the correct limitations on Router 2.

```
tc_delay = '10ms'
tc_rate = '70Mbit'
tc_bsize = '18750'

TPCONF_host_init_custom_cmds = {
   'pc02' : ['tc qdisc del dev enp13s1 root',
             'tc qdisc add dev enp13s1 root handle 2: netem delay %s' %
                 ↪ tc_delay,
             'tc qdisc add dev enp13s1 parent 2: handle 3: htb default
                 ↪ 10',
             'tc class add dev enp13s1 parent 3: classid 10 htb rate %s'
                 ↪  % tc_rate,
             'tc qdisc add dev enp13s1 parent 3:10 handle 11: bfifo
                 ↪ limit %s' % tc_bsize],
}
```

2.2 Traffic

In order for NETHINT to determine the rtt and owd for the network packets, it is necessary to use TCP. Between Server 2 and Client 2 we emulate VoIP traffic, for which we send 20 UDP packets per second with a packet size of 100 bytes (this mimics Skype, which will use TCP when UDP does not work and was found to send at roughly this rate and packet size with occasional outliers [4]). This is done using iperf with the flags `-b 16k -l 100 -i 0.05` at the client. This sends packets of size 100B every 0.05 seconds, which adds up to 16Kb per second. These flags are described in [5].

We run the tests with a few different types of traffic between Server 1 and Client 1. The different types of traffic are:

- 3 Reno
- 3 BBR
- 3 Cubic
- 1 Cubic + 1 BBR + 1 Reno
- 1 VoIP + 3 Reno

- 1 VoIP + 3 Cubic
- 1 VoIP + 3 BBR
- 1 VoIP + 1 Cubic + 1 BBR + 1 Reno

The VoIP traffic is set up the same way as between Server 2 and Client 2. The different TCP variants were also set up using iperf, but with the flag `-Z CCALGORITHM` where `CCALGORITHM` is one of reno, bbr, or cubic. We also initially had some tests for web traffic and VoIP traffic without the normal TCP traffic, but this did not produce enough traffic to cause any congestion.

3 Data

NETHINT produces JSON files with one JSON object for each data point, where the data points. The data points are not necessarily individual packets, but rather information concerning both the data packet and the corresponding ACK. These JSON objects include information such as the rtt and the owd for the packets.

References

[1] P. J. Barhaugen, "NETwork Home INTerference." May 08, 2023. Accessed: Nov. 30, 2023. [Online]. Available: https://github.com/petternett/NETHINT

[2] P. Juterud Barhaugen, "Home Network Interference Detection with Passive Measurements," University of Oslo, 2023.

[3] S. Zander and G. Armitage, "CAIA Testbed for TEACUP Experiments Version 2," 2015.

[4] M. Mazhar Rathore, A. Ahmad, A. Paul, and S. Rho, "Exploiting encrypted and tunneled multimedia calls in high-speed big data environment," *Multimed tools appl*, vol. 77, no. 4, pp. 4959–4984, Feb. 2018, doi: 10.1007/s11042-017-4393-7.

[5] "Manpage of IPERF." Accessed: Dec. 07, 2023. [Online]. Available: https://iperf2.sourceforge.io/iperf-manpage.html