Something

IN3260

Oskar Haukebøe

December 11, 2023

Contents

1 Introduction **1**

2 Testbed configuration **1**

2.1 Network Devices and Configuration 2

2.1.1 Configuration details 3

2.2 Examination of bottlenecks 3

2.2.1 Configuration of throughput limitations 4

2.2.2 Additional variable adjustments 4

2.2.3 TEACUP configuration for 4

2.2.4 Configuring Router 2 with `tc` 5

2.3 Traffic generation and types 5

3 Data **6**

1 Introduction

Network performance issues are a pervasive nuisance that can often leave users perplexed about their origin—be it within their local WiFi or due to external factors such as distant servers. Identifying these bottlenecks is critical in maintaining a proficient network experience.

One project that attempts to solve this challenge is NETHINT [1], which leverages passive WiFi traffic analysis to evaluate latency and packet loss, providing insights into whether the congestion lies in the local network or not.

This assignment builds upon the work of NETHINT by empirically testing its capability to discern the location of Bottlenecks within a controlled environment. Utilization of a TEACUP testbed [2] facilitates a systematic approach to simulating various network scenarios with different congestion points. Through automated experiments on real machines, this assignment aims to critically evaluate NETHINT's proficiency in accurately identifying whether local WiFi is the limiting factor. The TEACUP configuration used is available at [3].

2 Testbed configuration

The primary distinctive feature of NETHINT is its utilization of passive network measurements, ensuring no interference with the network [4]. It achieves this by monitoring

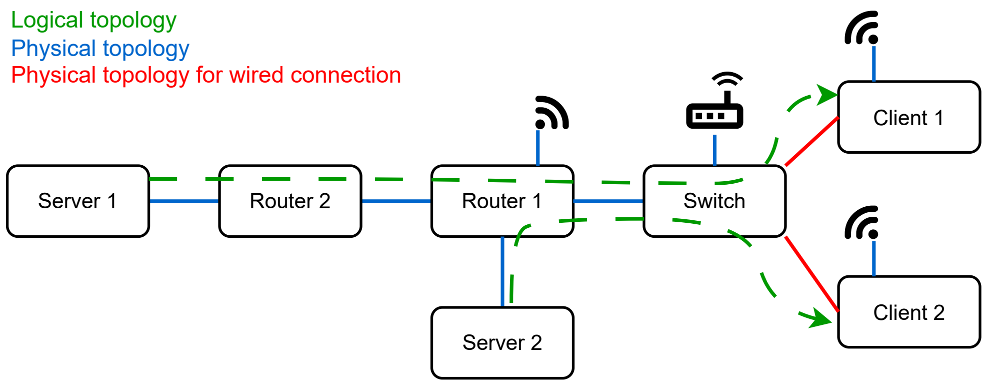

Figure 1: Topology of the testbed

WiFi traffic and analyzing packet timings for each device on the network. This allows for a comparative assessment of the packet delay across different devices, which then can be used to determine whether the devices on the network have a common bottleneck or not. Should a common bottleneck be identified, it likely indicates that congestion exists within the local area network (LAN).

To assess NETHINT's ability to recognize local WiFi bottlenecks, we configured a network topology as shown in Figure 1. All nodes in this topology are physical computers running Debian Linux, aside from the switch. They are managed by another computer not depicted in the figure, using TEACUP. This setup enables automated generation and control of network traffic throughput and delay at routers, as well as traffic logging.

2.1 Network Devices and Configuration

- *Servers:* Two servers (Server 1 and Server 2) for generating traffic.
- *Clients:* Two clients (Client 1 and Client 2) to act as traffic endpoints.
- *Routers:* Two routers with Router 1 being the entry point for both clients.
- *Switch:* A switch to connect the clients with Router 1, enabling bandwidth limitation at the router level.
- *WiFi AP:* An access point for the wireless segment of the network, set to channel 3 without encryption to accommodate NETHINT's capabilities.
- *Monitor Mode:* Router 1's WiFi card, set to monitor mode for traffic sniffing, is also configured to channel 3 to monitor the WiFi AP.

This setup models a scenario where Client 2 streams video from Server 2, while Client 1 consumes bandwidth for different activities from Server 1. The objective is to discern whether traffic between Server 1 and Client 1 interferes with the traffic between Server 2 and Client 2.

2.1.1 Configuration details

We use a switch to connect the two clients to the first router, allowing both clients to access Router 1 through the same network interface for wired connections. This setup facilitates bandwidth limitation at the router, enabling both clients to share the available bandwidth.

The WiFi AP is connected to the switch and provides an unencrypted WiFi network. The lack of encryption is due to NETHINT's inability to decrypt WiFi traffic. Both clients connect to this network, thus gaining access to the two servers. Router 1's WiFi card, configured in monitor mode, is employed by NETHINT to capture all the WiFi traffic between the clients and the WiFi AP. To enable traffic monitoring, the WiFi interface on Router 1 and the WiFi AP are both set to channel 3.

For the machines to recognize how to reach each other, they had to be configured to know where the packets should be sent. Client 2, for example, was configured in the file `/etc/network/interfaces.d/vlan11-iface` as follows:

```
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
```

The IP address `172.16.12.254` corresponds to the network interface of Router 1 that faces the switch. The address `172.16.11.0` is linked with Server 2, and `10.10.12.0` with Server 1. This configuration indicates to Client 2 that it can reach the servers via Router 1. The address `172.16.10.0` represents the interface on Router 1 facing Router 2 and was necessary for TEACUP operations, as it requires initial ping tests between certain nodes.

While the above outlines wired connectivity, the main tests needed to run wirelessly, hence the specification of `metric 10` in the routing commands. Given that clients can reach servers both via the WiFi AP and wired connections, it is crucial to ensure they utilize the correct network path. The following commands configure client preferences for network interfaces:

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

Here, `172.16.13.1` is the IP of the WiFi AP. These commands alternate the traffic route between the wired and wireless interfaces on the clients.

2.2 Examination of 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 occurs at Router 2, the user at Client 2 is not expected be significantly affected, as Router 1 should still have a surplus of bandwidth.

2.2.1 Configuration of throughput limitations

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. Depending on the desired bottleneck location, the throughput at Router 1 and Router 2 was adjusted either above or below this threshold, depending on where the bottleneck should be. Table 1 lists throughput values used in each bottleneck scenario.

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

2.2.2 Additional variable adjustments

In addition to throughput, bot queue length and delay were varied on Router 1. The queue lengths were set to:

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

Delays of 50ms and 10ms were tested at Router 1, with Router 2 consistently set at a 10ms delay.

2.2.3 TEACUP configuration for

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. Note that the delay here is set to 25ms, as TEACUP sets the delay for both downstream and upstream traffic.

While running the tests, TEACUP creates a matrix from these lists and executes the tests accordingly. However, this approach did not align with our objectives, as we intended to use distinct sets of buffer sizes for each delay configuration. Specifying multiple values for buffer sizes alone prevented an excessive number of test runs by limiting the variable combinations.

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

2.2.4 Configuring Router 2 with `tc`

To configure the delay for Router 2, we utilized the `tc` program to define qdisc rules. TEACUP provides a variable `TPCONF_host_init_custom_cmds` that executes specified commands on a designated machine. We used the following configuration to automatically enforce the desired constraints 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],
}
```

2.3 Traffic generation and types

For all tests, we use one VoIP traffic connection between Client 2 and Server 2, while we use a few different types of traffic between Server 1 and Client 1. We vary the type of TCP flows by changing the congestion control (CC) algorithm, and the type of traffic. These 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

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 [5]). 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 [6].

The configuration for VoIP traffic was consistent with that used between Server 2 and Client 2. Different TCP congestion control algorithms were configured in iperf with the `-Z` flag. We also initially had some tests with just web or VoIP traffic, without any normal iperf traffic, but this did not generate sufficient traffic to cause any congestion.

```
traffic_iperf = [
    # pc01 -> pc04
    ('0.0', '1', " start_iperf, client='pc04', server='pc01', port=5001,
        ↪ "
     " duration=V_duration, extra_params_client='-R',
         ↪ extra_params_server='-Z bbr' "),
    ('0.0', '2', " start_iperf, client='pc04', server='pc01', port=5002,
        ↪ "
     " duration=V_duration, extra_params_client='-R',
         ↪ extra_params_server='-Z bbr' "),
    ('0.0', '3', " start_iperf, client='pc04', server='pc01', port=5003,
        ↪ "
     " duration=V_duration, extra_params_client='-R',
         ↪ extra_params_server='-Z bbr' "),

    # pc03 -> pc05
    ('0.0', '5', " start_iperf, client='pc05', server='pc03', port=5001,
        ↪ "
     " duration=V_duration, extra_params_client='-b 16k -l 100 -i 0.05 -
         ↪ R' "),
]
```

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] S. Zander and G. Armitage, “CAIA Testbed for TEACUP Experiments Version 2,” 2015.

[3] Oskar, “Ohaukeboe/teacup-nethint.” Dec. 10, 2023. Accessed: Dec. 10, 2023. [Online]. Available: https://github.com/ohaukeboe/teacup-nethint

[4] P. Juterud Barhaugen, “Home Network Interference Detection with Passive Measurements,” University of Oslo, 2023.

[5] 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.

[6] “Manpage of IPERF.” Accessed: Dec. 07, 2023. [Online]. Available: `https://iperf2.sourceforge.io/iperf-manpage.html`