Supporting material for basic measurements

Markus Peuhkuri Seyud Mortezaei Sebastian Sonntag
Jiang Weixuan

Introduction

This supporting material will cover the main topics related to simple network measurements and footprinting by using simple measurements tools such as **ping**, **traceroute**, **curl**, **iperf3** and more. We will especially focus on latency measurement, throughput measurement, and finding available services.

After completion of this material the students have good understanding of the basic measurement and footprinting tools and how to use them in networks to get the basic performance measurement information.

Linux distributions do not have all of the programs pre-installed. In case of Ubuntu and other Debian-based distros, use the following command to install the packages (note: superuser privileges required, so you cannot use those in Aalto IT computers).

```
sudo apt-get install [PACKAGE]
```

This documentation complements Linux Introduction.

Discovering services

In general discovering other protocols and remote services can be accomplished in several different ways while in this example we mainly focus on port scanning methods for service discovery.

NMAP

NMAP is great software for discovering available services on remote machines by sending packets and analyzing their response. Then you are able to see what kind of services are available and then send suitable kind of packets to such destination. NMAP is free and available on both Windows and Linux platforms.

Table 1: NMAP options. Some scan types require elevated privileges.

Command	Description	Example
nmap x.x.x.x	Scan the remote host for open ports	nmap 192.168.100.250
nmap -v x.x.x.x	Verbose scan mode	nmap -v 192.168.100.250
nmap -sS x.x.x.x	SYN scan	nmap -sS 192.168.100.250
nmap -sA x.x.x.x	ACK scan	nmap -sA 192.168.100.250
nmap -F x.x.x.x	Fast scan	nmap -F 192.168.100.250
nmap -p y x.x.x.x	Specific port scan (TCP by default)	nmap -p 22 192.168.100.250
nmap -sU y x.x.x.x	Specific UDP port scan	nmap -sU 53 192.168.100.250
nmap -p y1- y2 x.x.x.x	TCP range port scan	nmap -p 1 - 99 20 192.168.100.250

For example, the command below shows all our machine's available TCP services and which ports they are using. You can compare the output to `ss -lt` command output.

```
nmap localhost
```

```
Starting Nmap 7.01 ( https://nmap.org ) at 2019-07-19 14:41 EEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000060s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 996 closed ports
PORT    STATE SERVICE
25/tcp  open  smtp
111/tcp open  rpcbind
222/tcp open  rsh-spx
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
```

Note that usually running `nmap` command may take a while and still miss services since it will scan by default 1000 most common ports of all 65,535 ports. A fast scan `-F` will scan only common 100 ports. If you want to scan all ports, option `-p-` can be used. Note that this will take a long time!

Also keep mind that scans may result complaints from network owners, so we recommend extensive scans only to hosts you have control on *or* are instructed to use.

Measuring round trip time (RTT), latency and connectivity

Next we will take a look some commands that could measure latency related things, such as **ping**, **dig**, **hping3**, **traceroute**, and **curl**.

PING

ICMP protocol in TCP/IP stack provides a simple and efficient mechanism to verify the connectivity, measure RTT, and packet loss rate among the different hosts in the network: `ping` command. When a host receives ICMP echo request message, it would respond to it by sending ICMP echo reply messages and copying data part of received message to replied message.

```
ping -c 3 www.iana.org
```

```
PING ianawww.vip.icann.org (192.0.32.8) 56(84) bytes of data.
64 bytes from www.iana.org (192.0.32.8): icmp_seq=1 ttl=244 time=190 ms
64 bytes from www.iana.org (192.0.32.8): icmp_seq=2 ttl=244 time=188 ms
64 bytes from www.iana.org (192.0.32.8): icmp_seq=3 ttl=244 time=189 ms
--- ianawww.vip.icann.org ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 188.360/189.579/190.660/0.944 ms
```

In example above, ping command was issued to send three ICMP echo requests to remote host `www.iana.org`. The `ping` command will print the result for each received reply message with useful details such as sequence number, TTL, time between request and reply messages, total number of transmitted/received packets, percentage of packet loss and also a summary of statistics including minimum, average, maximum, and mean deviation of RTTs.

Due to security reasons many hosts and web sites use firewall rules to drop ICMP messages which make it impossible to use ICMP protocol to get the RTT and connectivity check among the hosts. For example, the following attempt will result all packets lost:

```
ping -c 3 www.bbc.co.uk
```

```
PING gtmlivelive-eu-c-1nlbwwwcouk-17e0f31403454351.elb.eu-central-1.amazonaws.com (3.120.156

--- gtmlivelive-eu-c-1nlbwwwcouk-17e0f31403454351.elb.eu-central-1.amazonaws.com ping statis
3 packets transmitted, 0 received, 100% packet loss, time 2051ms
```

In these cases we need to use other methods and protocols to check the connectivity and RTT among the hosts, such as **dig** and **curl** commands.

Host and dig

Generally about DNS system, when you first time try to access some website with its domain name, your device must find out domain name's IP address before it can access the website. This is done by first sending DNS query to the configured DNS server which will then iteratively ask the target's IP address from various nameservers until it gets answer which is returned back to your device. Now your device can access the website which you typed initially, and the website's IP address will be stored in cache (which explains the next faster loadings for the website).

> If you are not familar with DNS system and do not know the difference beteen resolving and authoritative dns servers, recursion and iterative resolving. caches, domains and zones, we recommend you to to revisit some *Understanding DNS* course.

Command **host** is used to find IP-address of the host but it can also find other things such as mail servers for a domain. In some cases you need more detailed information to e.g. diagnose errors. The **dig** is the heavy-weight hammer to use.[1] The command performs DNS lookups and displays the answers that are returned from the nameserver(s) that were queried, plus the query time which could be useful for latency measurements.

Many DNS servers are only supposed to respond to DNS queries and not ICMP messages so you are not able to measure the latency or delay using ICMP messages. Instead, you could use `dig` command to see their response time. Next we query the root name servers with `dig command` (the command works via school servers but not from workstations).

```
dig fi @f.root-servers.net
```

```
; <<>> DiG 9.11.3-1ubuntu1.1-Ubuntu <<>> fi @f.root-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19507
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 9, ADDITIONAL: 19
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;fi.                IN  A

;; AUTHORITY SECTION:
fi.         172800  IN  NS  h.fi.
fi.         172800  IN  NS  b.fi.
fi.         172800  IN  NS  c.fi.
fi.         172800  IN  NS  d.fi.
fi.         172800  IN  NS  e.fi.
fi.         172800  IN  NS  f.fi.
fi.         172800  IN  NS  g.fi.
fi.         172800  IN  NS  a.fi.
fi.         172800  IN  NS  i.fi.

;; ADDITIONAL SECTION:
h.fi.           172800  IN  A   87.239.120.11
```

[1] Dig command may not be installed by default. It is in `dnsutils` package in Ubuntu/Debian.

```
h.fi.           172800  IN  AAAA    2001:678:a0::aaaa
b.fi.           172800  IN  A   194.146.106.26
b.fi.           172800  IN  AAAA    2001:67c:1010:6::53
c.fi.           172800  IN  A   194.0.11.104
c.fi.           172800  IN  AAAA    2001:678:e:104::53
d.fi.           172800  IN  A   77.72.229.253
d.fi.           172800  IN  AAAA    2a01:3f0:0:302::53
e.fi.           172800  IN  A   194.0.1.14
e.fi.           172800  IN  AAAA    2001:678:4::e
f.fi.           172800  IN  A   87.239.127.198
f.fi.           172800  IN  AAAA    2a00:13f0:0:3::aaaa
g.fi.           172800  IN  A   204.61.216.98
g.fi.           172800  IN  AAAA    2001:500:14:6098:ad::1
a.fi.           172800  IN  A   193.166.4.1
a.fi.           172800  IN  AAAA    2001:708:10:53::53
i.fi.           172800  IN  A   162.88.44.1
i.fi.           172800  IN  AAAA    2600:2000:3008::1

;; Query time: 6 msec
;; SERVER: 2001:500:2f::f#53(2001:500:2f::f)
;; WHEN: Wed Sep 12 20:03:56 EEST 2018
;; MSG SIZE  rcvd: 571
```

As an example to this case, when we query for available nameservers responsible for serving `.fi` ccTLD, we will get a query reply in 80ms which is quite close to what has been provided by traceroute. This is not always a case.

However for this example we will deal with root servers and as root servers are serving only name service info for top level domains (gTLD like `.com`, `.net` and ccTLD like `.fi` or `.it` and don't know nothing else about particular branching domains like `example.com` or `example.fi`, response times can be used as an estimation for the delay when the ICMP protocol is blocked by firewall rules.

One very command issue with DNS queries are the caching servers. By default, both the **host** and **dig** will use the DNS server configured into system (e.g. received via DHCP request). These name servers are *resolving* name servers that will *cache* the result.

However, because of the cache memory, continuosly repeating dig command will result in zero latency which is misleading. You can easily test this with following commands:

```
#!/bin/bash
# generate random domain, most likely
# 1) would not exists
# 2) nobody has ever asked for that domain
# name has about 310 bits of entropy
dom=$(pwgen -sn 60 1).com
```

```
echo Asking for host: $dom
dig $dom | fgrep time:
dig $dom | fgrep time:
```

Above script will ask for a random domain name twice in sequence. The local name server will cache even negative response for a short period of time. The second time should be a significantly shorter as it will be answered directly from the local name server.

If the script is modified so that no local server is used but the authoritative one, the responses should be more or less the same.

```
# other lines as before
dig +norec $dom @k.gtld-servers.net | fgrep time:
dig +norec $dom @k.gtld-servers.net | fgrep time:
```

If you want to measure only network latency, make sure you ask only such information that the server can reply right away. Using `+norec` flag instructs name server not to initiate recursion, so the answer will come from local database or local cache from server.

HPING and TCPPING

After detecting other services on remote machine we are able to use some special tools such as `hping3` or `tcpping` (`tcping` in Windows) to check the connectivity and RTT. For example, `hping3` is special ping-like command where the packet can be customized. For instance, you are able to modify its real source IP-address or give it TCP SYN flags. This is especially useful for testing own firewall systems. However, notice that the latency of `hping3` can be higher since its protocol is TCP/IP which is not as lightweight as ICMP protocol. Note that these may require super user access to run thus one cannot use them on Aalto computers.

```
$ sudo hping3 -c 3 -S -p 80 www.aalto.fi
HPING www.aalto.fi (wlp2s0 192.230.77.222): S set, 40 headers + 0 data bytes
len=46 ip=192.230.77.222 ttl=56 DF id=0 sport=80 flags=SA seq=0 win=29200 rtt=31.7 ms
len=46 ip=192.230.77.222 ttl=56 DF id=0 sport=80 flags=SA seq=1 win=29200 rtt=31.4 ms
len=46 ip=192.230.77.222 ttl=56 DF id=0 sport=80 flags=SA seq=2 win=29200 rtt=31.5 ms

--- www.aalto.fi hping statistic ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 31.4/31.5/31.7 ms
```

Above `hping3` command will issue three TCP (SYN) ping to destination www.aalto.fi port 80.

Now despite of dropping of ICMP packets by the firewall rules we are still able to check the connectivity and RTT information.

Traceroute

There are some situations where the network problem is not in the source or destination host but in the path between the source and destination. `traceroute` is another basic tool which can be used to detect all the hops among the source and destination hosts, and also the RTT between the hops.

```
$ traceroute www.mtv3.fi
traceroute to www.mtv3.fi (178.217.129.234), 30 hops max, 60 byte packets
 1  jgw-2-v834.aalto.fi (130.233.224.195)  0.220 ms  0.218 ms  0.246 ms
 2  funet-10g-aalto-a.aalto.fi (130.233.231.189)  1.147 ms  1.167 ms  1.160 ms
 3  espoo1-et-0-1-6-1.ip.funet.fi (86.50.255.253)  0.559 ms  0.575 ms  0.566 ms
 4  espoo2-et-0-1-7-1.ip.funet.fi (86.50.255.233)  1.068 ms  1.065 ms  1.059 ms
 5  tut6-et-0-1-0-1.ip.funet.fi (193.166.255.51)  2.994 ms  3.007 ms  3.022 ms
 6  cybercom1.unicast.trex.fi (195.140.192.20)  3.725 ms  3.544 ms  3.526 ms
 7  cdn-vip27.hard.ware.fi (178.217.129.234)  3.550 ms  3.591 ms  3.557 ms
```

And like before, when the ICMP protocol is blocked by firewall rules we can use `tcptraceroute` if permitted or we have root access.

So far we got familiar with both ping and traceroute programs. The **mtr** (**pathping** in Windows) is another useful command which combines the functionality of the traceroute and ping programs in a single network diagnostic tool.

Latency measurement with curl

Some of tools listed above are not possible to use with Aalto generic computers as they require elevated privileges to send crafted packets. However, it is still possible to measure RTT using TCP connection establishment delay as measure. You can create your own program using C or python, for example, but the Swiss knife of network transfers, curl provides an easy option.

Following script will connect to a site given as an argument and prints out CSV with columns time since starting from initiating the request.

1. Reply from DNS lookup was received.
2. TCP connect was completed.
3. Transfer started.
4. Transfer completed.

```
#!/bin/bash
export LANG=C
fmt="%{time_namelookup}, %{time_connect}, %{time_starttransfer}, %{time_total}\n"
curl -w "$fmt" -o /dev/null -s ${1:-http://www.aalto.fi}
```

Time given is cumulative, i.e. to get TCP connect time you need to subtract DNS lookup. As we want only the CSV output, the content of file is send to bottomless sink `-o /dev/null` and the progress indicator is silenced `-s`.

The `export LANG=C` line above will make sure the output will use period as decimal separator. If your user session default language is Finnish, for example, the decimal separator is comma. That will result problems if the column separator is also comma. The other alternative is to use for example semicolon `;` or tabulator between columns.

You may also want to check other `--write-out` information variables from curl documentation from web or manual page. Make note to check *which features are available* in the version you are using.

Measuring network throughput

Most network users are interested only about a single factor: how many bits per second one can download or send with this network connection. This is also what is advertised by network providers. Here three simple ways to measure throughput are shown: *file transfer*, *dedicated bandwidth tools*, and *measurement services*.

File transfer as test

A simple test is to measure how long it takes to transfer a large file. We can then get the throughput by dividing the file size with the time it took to download. You can use standard browser and stopwatch for this, with possibility to use developer tools for more detailed output.

For example, `ftp.funet.fi` has included several files in /dev directory. There are both null files that contain just NUL characters (byte value 0) and `rnd` where the content is random bytes. There are several commands to download such files such as by using **curl**, **wget**, and **ftp** commands.

Here is quick example of downloading 100 MiB of random data with curl from Funet.

```
curl -o /dev/null ftp://ftp.funet.fi/dev/100Mrnd
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  100M  100  100M    0     0  5542k      0  0:00:18  0:00:18 --:--:-- 5571k
```

The output above displays that the average download speed was 5,542 kibibytes/s. For more easy-to-process output check curl `--write-out` documentation above. If we use the `curl-latency.sh` script from latency measurements, we will get following output that actually gives more resolution to the elapsed times. Maybe add few variables like `speed_download` and `size_download` for printout to get throughput easy?

```
0.024805, 0.045800, 0.067767, 18.558943
```

In addition, a compression test using bzip2 for 100M_type_ from Funet resulted the original 104,857,600 byte *null* file to become a 5615 byte file from and

the *rnd* to a 103,631,335 byte file. The *null* file contains only single character (NUL) while the *rnd* contains more random data that barely compress any. Comparing different payload download times can be used to study if there is any data compression on link. As much of internet traffic is now encrypted, link compression has very little value.

Dedicated bandwidth tools

A second way for measuring the actual throughput is to use special tool like **iperf3** that allows more control on file transmission and it is not dependent on any file or storage like file transfer by FTP. Notice that this command works in client-server fashion, so the destination must be running iperf3 in server mode to get the result. There are many public iperf-servers available for testing, Funet server is one of them. By default `iperf3` only sends data towards server but it also supports reverse tests.

The `iperf3` supports following command lines:

- `-c host`: Use server host (IP address or host name)
- `-t time`: Run test `time` seconds, by default 10 seconds.
- `-R`: Run test reverse (the server will send data)
- `-w size`: Specify TCP window size (like 128k for 128 kilobytes).
- `-J`: output results as JSON format for more detailed information.
- `-s`: Run as server. Exclusive for -c option

If you are using `iperf3` on Windows host, note that there are multiple *"Iperf Windows"* or *"GUI Iperf"* versions available where some of them are commercial softwares.

Also note that version 3 (`iperf3`) is not compatible with version 2 (`iperf`) servers. Basic usage is still similar. You can use either version, but the version 3 works for download tests or when connections are made behind firewall or NAT.

```
iperf3 -c iperf.funet.fi
```

```
Connecting to host iperf.funet.fi, port 5201
[  4] local 2a00:1190:c005:1000:7429:f4ac:fe12:5fba port 54262 connected to 2001:708:0:3::2
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  1.04 MBytes  8.72 Mbits/sec   19   20.9 KBytes
[  4]   1.00-2.00   sec   251 KBytes  2.06 Mbits/sec    0   23.7 KBytes
[  4]   2.00-3.00   sec   251 KBytes  2.06 Mbits/sec    1   23.7 KBytes
[  4]   3.00-4.00   sec   251 KBytes  2.06 Mbits/sec    2   13.9 KBytes
[  4]   4.00-5.00   sec   251 KBytes  2.06 Mbits/sec    0   22.3 KBytes
[  4]   5.00-6.00   sec   251 KBytes  2.06 Mbits/sec    1   22.3 KBytes
[  4]   6.00-7.00   sec   251 KBytes  2.06 Mbits/sec    2   22.3 KBytes
[  4]   7.00-8.00   sec   251 KBytes  2.06 Mbits/sec    1   20.9 KBytes
[  4]   8.00-9.00   sec   251 KBytes  2.06 Mbits/sec    1   20.9 KBytes
[  4]   9.00-10.00  sec   251 KBytes  2.06 Mbits/sec    1   22.3 KBytes
```

```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  3.25 MBytes  2.72 Mbits/sec   28             sender
[  4]   0.00-10.00  sec  2.42 MBytes  2.03 Mbits/sec                  receiver

iperf Done.
```

From the result above, we can see that the command possibly measures several maximum throughputs which were achieved, and at the end the average throughput is then displayed. In addition, retransmissions are also taken into account when calculating the average throughput.

Measurement as service

In case where someone cannot afford having a Linux host with gigabit links to run `iperf`, there are sites that provide measurement service. These websites includes Nettitutka developed at Aalto University and Speedtest. Nettitutka is focused on mobile devices and geographical information on mobile networks while Speedtest supports also desktop browsers. Both services also provides latency information.