Networking Lab 6 – Performance test on Ethernet links

The goal of this laboratory is to evaluate the performance of a file transfer over the network and observe the impact of different mechanisms on the performance. The focus is on performance obtained using TCP or UDP at the transport layer, considering scenarios of increasing complexity.

To measure the performance, we will use either the `nttcp` or the `iperf` command, described in the next sections.

We are interested in measuring what we will refer to as the "**goodput**", that is, the speed at which *useful* data is received at the *application* layer.

Goodput = Useful **data** at the application layer / **time** to complete the transfer

This is different from the **throughput**, that is, the amount of information carried by the physical layer. The latter includes all protocol overheads, e.g., Protocol Control Information added by each layer, retransmission due to error recovery mechanisms implemented at layer 2 or layer 4, interfering traffic due to other applications sharing the link such as ARP requests, etc.

1. What is the maximum goodput you expect when using UDP at the transport layer? Compute the maximum efficiency μ_{UDP} defined as the *per packet goodput*, i.e., the fraction of useful data carried by a UDP message with respect to physical layer bits being transmitted.
2. What is the maximum goodput you expect when using TCP at the transport layer? Compute the maximum efficiency μ_{TCP} defined as the *per packet goodput,* i.e., the fraction of useful data carried by a TCP segment with respect to physical layer bits being exchanged. The latter includes all headers, acknowledgements too.
3. What changes if the link is set to HALF DUPLEX? Compute the maximum efficiency $\mu_{TCP\text{-}HD}$.

To allow a better control of the test bed, configure your Ethernet interface so that

- All **offloading** capabilities are **disabled**
- The speed is **10Mb/**s Full Duplex unless otherwise specified.

```
H1: ethtool -s eth1 speed 10 duplex full autoneg on
```

NOTE: the NIC may be also support the Ethernet flow control mechanism known and the **PAUSE Frame** protocol defined in the IEEE 802.3x standard. It allows the switch to send a PAUSE Frame which inhibits the transmission of further frames on the other transmitter side. This avoids congesting the switch queue by blocking the upstream sender. Try to check and eventually disable it using the ethtool

```
H1: ethtool -a eth1 # shows the status of the pause support
```

```
H1: ethtool -A eth1 [autoneg on|off] [rx on|off] [tx on|off]
```

Warning: this setting is critical, and not all line cards allow you to control the configuration in a proper way. Be careful.

4. nttcp

The `nttcp` application measures the transfer rate (and other indexes) on a TCP, UDP or UDP multicast connection. To use `nttcp` you must run the application on the local host (client) and on the remote host (server). On the server host simply start `nttcp` with the option `-i`.

```
H1: nttcp -i &
```

[note: the `&` runs the application in background so that the terminal is not blocked waiting for the application to exit].

Alternatively, you may wish to configure the `/etc/inetd.conf` – see the `nttcp` and `inetd` manpage

On `H1, nttcp` is now listening for connections from client `nttcp` applications.

> Which port is the `nttcp` server listening to?

On the client H2 simply call `nttcp` with the name of the partner host.

```
H2: nttcp H1
```

H2 will contact the `nttcp` server running on the H1 and it will initiate the test. By default, the test transfers 2048 buffers of 4KByte length (a total of 8 MByte) to the partner host. The goodput performance will be measured on both sides, and the results (both local and remote) are reported on the client terminal as output. You can change parameters of the transmission via command line options.

Useful options:

`-r` defines the **receive** transfer direction; data is sent from the partner host to the local host.

`-t` defines the **transmit** transfer direction; data is sent from the local host to the partner host. This is the default direction.

`-T` Print **a title line**.

`-u` Use the **UDP** protocol instead of TCP (which is the default).

`-n N` The **number N of buffers** will be written to the transmitting socket. It defaults to 2048.

`-l L` The length `L` defines the **size of one buffer** written to the transmitting socket. Defaults to 4096.

For example, let H2 receive 100.000 UDP frames with a payload of 100B from H1:

```
H2: nttcp -r -u -n 100000 -l 100 H1
```

> 1. What is the impact of the choice of the N and L parameters? Does it change if you use TCP or UDP? Why?
> 2. How should you choose N and L to get reliable measurements? Remember: `nttcp` is measuring the **time** spend to complete the transfer. How long should be the test to avoid eventual bias due to errors in measuring the duration of the transfer?
> 3. If you repeat the measurement several times, do you get the same numbers? If not, why?
> 4. Are the measurements reported by the local host and the remote host different? Why?
> 5. Does it change if H1 transmits data to H2 or viceversa? Why?
> 6. How can `nttcp` measure the time spent to *complete* the transfer? Does TCP or UDP change the scenario? Which assumption are required?

5. Iperf

`iperf` is another tool for performing network throughput measurements. It can test either TCP or UDP goodput. As for `nttcp`, to perform an `iperf` test, the user must establish both a server (to discard traffic) and a client (to generate traffic).

To setup a server, run `iperf` as below

```
H1: iperf -s
```

As in the case of `nttcp`, now an `iperf` server is running on H1, and waiting for incoming connections.

1. Which port is the `iperf` server listening to?
2. Is it possible to have an `iperf` UDP and TCP server, and a `nttcp` server at the same time?

On the client H2 simply call `iperf` with the name of the partner host.

```
H2: iperf -c H1
```

This will run a test, using TCP at the transport layer, with `H2` sending data to `H1`. The test will last 10s. This is different from `nttcp`, where you have the control on the amount of data to be sent, with `iperf` you have the control on the time the test lasts. Note that the `iperf` client will act as the sender by default.

Useful option:

GENERAL OPTIONS

`-i, --interval n`: pause n seconds between periodic bandwidth reports

`-o, --output <filename>`: output the report or error message to this specified file

`-u, --udp`: use UDP rather than TCP

`-w, --window n[KM]`: TCP window size (socket buffer size)

`-M, --mss n`: set TCP maximum segment size (MTU - 40 bytes)

SERVER SPECIFIC OPTIONS

`-s, --server`: run in server mode

CLIENT SPECIFIC OPTIONS

`-b, --bandwidth n[KM]`: set target bandwidth to n bits/sec (default 1 Mbit/sec). This setting requires UDP

`-c, --client <host>`: run in client mode, connecting to <host>

`-d, --dualtest`: Do a bidirectional test simultaneously

`-r, --tradeoff`: Do a bidirectional test individually

`-t, --time n`: time in seconds to transmit for (default 10 secs)

`-P, --parallel n:` number of parallel client threads to run

`-Z, --linux-congestion <algo>:` set TCP congestion control algorithm (Linux only)

`Iperf` offer some nice possibilities which will allow us to have more control on the experiments. In particular, `iperf` allows one to choose the TCP congestion control the sender will use by specifying it via the `–Z` option. Linux support several algorithms, among which:

- Original additive Increase, multiplicative decrease version: `Reno`
- Versions designed for large bandwidth delay product paths: `Bic, Cubic, Highspeed, Htcp, Illinois, Scalable`
- Versions designed for satellite links: `Hybla`
- Version that support low priority traffic: `lp`
- Congestion control driven by delay instead of losses, or that specifically are designed for wireless links: `Vegas`, `Veno`, `Westwood`, `yeah`

See http://interstream.com/node/1084 or http://linuxgazette.net/135/pfeiffer.html for more details.

Similarly, `iperf` allows one to control the TCP window size via the `–w` parameter, and to eventually use multiple parallel connections via the `–P` option. NOTE: the actual value of the window size is typically different from the one given as input – always double check which is the actual value when running an experiment.

6. Computing the goodput

Consider now the following different scenarios, with increasing complexity. For each of them

- **Predict the goodput you expect** when TCP or UDP is used at the transport layer
- In scenario where TCP and UDP flow coexists, predict the goodput you expect for each flow
- Is it possible to observe **collision** at the Data Link Layer? If so, which would be the collision probability? Suggestion: check the number of collision before and after the test using the `ifconfig` command.
- Is it possible that **frames** are **dropped** by the switch? You could compare the number of frames sent on the sender NIC with the number of frames received on the receiver NIC via `ifconfig`.
- Is it possible that **data is lost** in the protocol stack?

After having discussed the previous questions, run the actual test and check your predictions.

How good was it? Comment on eventual disagreements. Why you got something different from what you were expecting?

A. Single flow – FULL DUPLEX: H1 is sending data to H2
B. Single flow – HALF DUPLEX: H1 is sending data to H2, but the linecard of H1 is set to half duplex. Does it change if H1 receives data from H2? Why? What if both linecards are in half duplex mode?
C. Bidirectional flows – FULL DUPLEX: H1 is sending data to H2 *AND* H2 is sending data to H1. To allow "concurrent" flows, you can concatenate two `nttcp` commands, like `H1: nttcp -T H2 >tx.dat & nttcp -T -r H2 >rx.dat`, or you can use `iperf -d` option
D. Bidirectional flows – Half DUPLEX: H1 is sending data to H2 AND H2 is sending data to H1. Linecard of H1 is in half duplex mode. What if other cards are in half duplex too?

E. Two flows to the same *receiver* – FULL DUPLEX: H1 sends data to H2 *AND* H3 sends data to H2
F. Two flows to the same *receiver* – HALF DUPLEX: H1 sends data to H2 *AND* H3 sends data to H2
G. Two flows from the same *sender* – FULL DUPLEX: H1 sends data to H2 *AND* to H3
H. Two flows from the same *sender* – HALF DUPLEX: H1 sends data to H2 *AND* to H3
I. Full mesh scenario- FULL DUPLEX: Each hosts sends data to all hosts at the same time.
J. Full mesh scenario- HALF DUPLEX: Each hosts sends data to all hosts at the same time.

Summarize results in the tables below:

Test	Average TCP Goodput per flow		Collision probability		Loss at the application layer		Comment
	Prediction	Observed	Prediction	Observed	Prediction	Observed	
A							
B							
C							
D							
E							
F							
G							
H							
I							
J							

Table 1: Summary of TCP goodput experiments

Test	Average UDP Goodput per flow		Collision probability		Loss at the application layer		Comment
	Prediction	Observed	Prediction	Observed	Prediction	Observed	
A							
B							
C							
D							
E							
F							
G							
H							
I							
J							

Table 2: Summary of UDP goodput experiments

Test	Average TCP Goodput per flow		Average UDP Goodput per flow		Comment
	Prediction	Observed	Prediction	Observed	
C					
D					
E					
F					
G					
H					
I					
J					

Table 3: Summary of mixed TCP and UDP goodput experiments