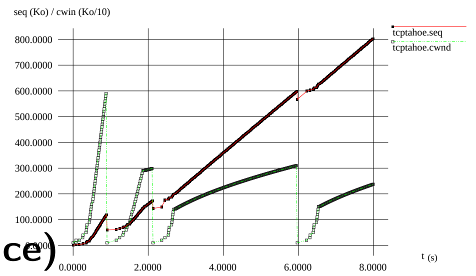
ComNet - Lab n°5

Transport Layer (2) : TCP Congestion Control



1 Congestion Control (without computer assistance)

TCP is used for reliable transport of data in the Internet. We previously study connection management and TCP mechanisms. In the following exercises, we will get interest in a other fundamental behavior of TCP: the congestion control.

1.1 Congestion detection

TCP was designed at the end of the 70's. Several congestion control algorithms have been added since, mainly following the work of Van Jacobson published in 1988. They continue to evolve in different TCP variants. The exercises proposed in the following are founded on the last versions: RFC 5681 of September 2009.

1. For TCP, which phenomenon indicates congestion in the network?
2. What's going on inside a router to generate this phenomenon?
3. For TCP, this phenomenon can infer congestion. But it can also occur when there is no congestion in the network. In which other cases in which case such a phenomenon may occur?
4. If this phenomenon does not always indicate congestion, why is TCP based on this inference? Why don't we use an approach where the router notify explicitly the congestion by sending a message to the sender?

1.2 Congestion Control Algorithms

For congestion control, TCP uses a threshold that indicates the flow rate above which congestion may occur. This threshold is expressed by the parameter `ssthresh` (in bytes). To get the flow rate threshold, `ssthresh` is divided by the RTT (Round Trip Time). The flow rate can vary from below and above the threshold `ssthresh`$/RTT$. The issuer maintains an other parameter, `cwnd` (Size of the congestion window), which indicates the maximum number of bytes it can send before receiving an acknowledgment. When `cwnd > ssthresh`, the sender take care particularly to not cause congestion.

1. Suppose `ssthresh` is at 5000 bytes, `cwnd` is at 6000 bytes, and segment size is 500 bytes. The sender sends twelve segments of 500 bytes in one RTT period, and receives twelve acknowledgements (one for each segments). What happens to the values ??`ssthresh` and `cwnd`? How these values changes are called?
2. Suppose `ssthresh` is still at 5000 bytes, `cwnd` is now at 14,000 bytes, the sender sends $14.000/500 = 28$ segments, and that the sender receives a congestion notification before receiving the first acknowledgement. What happens to the `ssthresh` and `cwnd` values??? How these values changes are called?
3. We have seen how increases and decreases `cwnd` depending on the absence or presence of indicators of congestion. How do we call this algorithm? On what principle is based this algorithm?
4. At startup, and after having received a congestion notification, the value of `cwnd` is smaller than the value of `ssthresh`. Describe how `cwnd` increase when it is lower than `ssthresh`, depending on the following example. Suppose `ssthresh` equal to 3000 bytes and `cwnd` equal to 500 bytes, the size of a segment. The transmitter has several segments ready to be sent. How many segments sends the issuer during the first RTT period? If it receives acknowledgments for all segments, what becomes the value of `cwnd`? How many segments sends the issuer during the second RTT period? If it receives acknowledgments for all segments, what becomes the `cwnd` value? In general, how evoluate the size of `cwnd`?

5. How is called the period during which `cwnd` is smaller than `ssthresh`?
6. What happens to the value `ssthresh` if the sender receives a congestion notification while `cwnd` is smaller `ssthresh` that?

1.3 Average bandwidth of a TCP connection

Suppose we wish to perform a large data transfer through a TCP connection

1. By neglecting the period during which `cwnd` is smaller than `ssthresh`, show that d, the average flow rate associates to a TCP connexion, is equal to:

$$d = \frac{3}{4}\frac{W * MSS}{RTT}$$

   where W is the size of the window (in segments) at the time of congestion, MSS the size of segment (assumed to maximum), and RTT is the round trip delay (assumed constant during the period of transmission).

2. Show that the loss rate p is equal to:

$$p = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

3. Show that if the loss rate observed by a TCP connection is p, then d, the average flow rate, could be approximated by:

$$d = \frac{1,22 * MSS}{RTT\sqrt{p}}$$

4. What other parameters can affect the throughput of a TCP connection?
5. What utility do you see to the relation calculated in the last formula of d?

2 Study of the latency of a web server (without computer assistance)

We would like to study the latency bound to the answer to an HTTP request [1] We make the following simplifying assumptions:

- The network is not congested (no losses or retransmissions);
- The receiver has infinite reception buffer (transmitter only limitation is due to the congestion window);
- The size of the object to receive the server is O, an integer multiple of MSS (MSS size is S bits);
- Throughput of the link connecting the client to the server is R (bps) and we neglected the size of all headers (TCP, IP and Link layer). Only the segments carrying data have a significant transmission time. The transmission time of control segments (ACK, SYN ...) is negligible;
- The value of the initial threshold of congestion control is never reached;
- The value of the round trip time delay is RTT.

1. Initially, we assume that we have no congestion control window. In this case, justify the following expression of L, the latency:

$$L = 2RTT + O/R$$

2. We now assume a **static** congestion window of fixed size fixe W. Calculate the latency in this first case:

$$WS/R < RTT + S/R$$

[1] Latency of an HTTP request: the time for creating the connection and the receiving of the complete application object.

3. We still assume a **static** congestion window of fixed size W. Calculate the latency in the second cases:

$$WS/R > RTT + S/R$$

4. Compare the latency with a **dynamic** congestion control window (slow start) with the one without congestion control.

5. Numerical implementation:

R	O/R	L (without slow start)	K'	L (TCP global latency)
56 Kbps				
512 Kbps				
8 Mbps				
100 Mbps				

K' is the number of windows sent before starting the second case $(log2(1 + RTT * R/S))$. Consider three cases:

(a) $S=$ 512 octets, $RTT=$ 100 msec, $O=$**100 Koctets** (=200S);

(b) $S=$ 512 octets, $RTT=$ 100 msec, $O=$**5 Koctets** (=10S);

(c) $S=$ 512 octets, $RTT=$ **1 seconde**, $O=$5 Koctets (=10S).

3 Analysis of TCP mechanisms

In the following, we analyse HTTP traffic captures. Depending of the availability of the EdgeNet testbed, you will capture by yourself long distance traffic or you will use the first three following capture pre-recorded on long distance client and server. (All three are made using a probe close to the client). In all cases, you will realize by yourself the last one in a local environment (on the networking testbed). For each one, draw the chronogram and study the congestion control mechanisms put-in-work. Discuss particular of the following:

1. What is the average RTT?
2. Do you recognize the congestion control mechanisms?
3. Up to how many segments are transmitted by RTT?
4. What is the average throughput achieved then?
5. A continuous sending it appears?
6. Is their any disturbances (desequencement, retransmission ...)?

3.1 HTTP Traffic Paris-Brisbane (WAN Intercontinental: 17000km)

One your PPTI host, use the `wireshark` tool (without needing administrator rights) for this first trace. Download the trace `tme5-wau.dmp` either from the directory `/Infos/lmd/2022/master/ue/MU4IN001-2022oct`, or from the web page `http://www-npa.lip6.fr/~fourmaux/Traces/labV8.html`.

3.2 HTTP Trafficrafic HTTP Paris-Budapest (WAN Continental : 1200km)

One your PPTI host, use the `wireshark` tool (without needing administrator rights) for this trace. Download the trace `tme5-whu.dmp` from the previous locations.

3.3 HTTP Trafficrafic HTTP Paris-Evry (MAN : 36km)

One your PPTI host, use the `wireshark` tool (without needing administrator rights) for this trace. Download the last trace `tme5-man.dmp` from the previous locations.

3.4 Local HTTP Traffic (LAN)

3.4.1 Capturing TCP traffic resulting from local HTTP exchange

First, you will capture some TCP traffic inside a LAN in order to understand its basic characteristics. Topology 1 (with client and server on the same LAN, as described in Lab n°1) is available to you on the networking testbed. Capture traffic using `wireshark`, as follows:

- From PPTI PC **N**, connect to the three corresponding VMs of the testbed from three separate terminal windows
  - in the "client" vm**N**1 (window 1), type: `ssh -Y etudiant@10.0.7.`**N**`1`
  - in the "monitor" vm**N**2 (window 2), type: `ssh -Y etudiant@10.0.7.`**N**`2`
  - in the "server" vm**N**3 (window 3), type: `ssh -Y etudiant@10.0.7.`**N**`3`
- Verify a large file is reachable from the directory `public_html` of account `etudiant` (to create with public access rights) and the HTTP server is running on VM `10.0.7.`**N**`3` (windows 3)
  - generate a large file, type: `dd if=/dev/zero of=public_html/fichier100Mo bs=1M count=100`
  - verify the public access of the file(`-rw-r--r--`), type: `ls -l public_html/fichier100Mo`
  - look for the HTTP server process (`apache2`)
  - query the VM about its network interfaces, look at the information regarding the interface to the experimental LAN and verify that its IP address is `10.`**N**`.1.`**N**`3`
- Start the capture by running the sniffer on the monitor VM (window 2)
  - run the sniffer by typing: `wireshark`
  - initiate the capture on interface `eth1`, as described in Lab n°1
- Start a HTTP client on the client VM (window 1)
  - start the HTTP client of your choice (`firefox`...)
  - type the following URL `http://10.`**N**`.1.`**N**`3/~etudiant/fichier100Mo`
- Observe the trace being captured in the `wireshark` window
- Complete the capture, then **filter the traffic to keep only TCP** (filter = `tcp`). Save the filtered trace for later reuse. Keep the application running and answer to the same questions than for the three previous traces.

3.4.2 Without the testbed...

If you have difficulty accessing the networking testbed, or you would simply prefer to work from another machine, you can download the trace `tme5-lan.dmp` (similar to the one previously captured) either from the directory `/Infos/lmd/2022/master/ue/MU4IN001-2022oct`, or from the web page `http://www-npa.lip6.fr/~fourmaux/Traces/labV8.html`, and then analyze it with `wireshark` (without needing administrator rights).

4 Before leaving the room

- If you have saved some traces on the monitor VM, do not forget to transfer them back to your PPTI user account. Type the following command on a local terminal of the PPTI host: `scp etudiant@10.0.7.`**N**`2:<trace> <dest>`
- Before closing your connections to the virtual machines, be sure to restore them to the state in which you found them, removing any modifications you might have made.