Computer Networks - Lecture 9 & 10

- Author: Ruben Schenk
- Date: 01.06.2021
- Contact: ruben.schenk@inf.ethz.ch

3.3.6 Timeouts & Retransmissions

TCP resets a timer whenever new data is ACKed, not on a duplicate ACK. Upon a timeout, the packet containing the next byte, i.e., the seqNo of the ACK packet.

The problem here is that setting the timeout value is hard. Choosing it too short will result in duplicate packets, choosing it too long will result in inefficiencies. In order to set the timeout value, we need some estimate of the connections RTT.

> The `Kern-Partridge Algorithm` gives us an idea on how to estimate the RTT. We start with measuring the $\mathrm{SampleRTT}$ for original transmissions, that is, not for retransmissions. We are then able to estimate the RTT the following way:
>
> $$\mathrm{SampleRTT} = \mathrm{AckRcvdTime} - \mathrm{SendPacketTime}$$
> $$\mathrm{EstimatedRTT} = \alpha \times \mathrm{EstimatedRTT} + (1-\alpha) \times \mathrm{SampleRTT}$$
>
> for $\alpha = 0.875$. From this, we determine the `retransmission timeout (RTO)` by
>
> $$\mathrm{RTO} = 2 \times \mathrm{EstimatedRTT}$$
>
> We might use an `exponential backoff`, that is, upon a timeout we double the RTO. Every time a new measurement comes if, we collapse the RTO back to $2 \times \mathrm{EstimatedRTT}$.

3.3.7 Congestion Control

`Congestion control` aims at solving three problems:

- *Bandwidth estimation*: How to adjust the bandwidth of a single flow to the bottleneck bandwidth?
- *Bandwidth adaptation*: How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
- *Fairness*: How to share bandwidth fairly among flows, without overloading the network?

It is important to note that congestion control differs from flow control, but botha re provided by TCP though:

- `Flow control`: prevents one fast sender from overloading a slow receiver.
- `Congestion control`: Prevents a set of senders from overloading the network.

The sender adapts its sending rate based on these two windows:

- `Receiving Window (RWND)`: How many bytes can be sent without overflowing the receiver buffer?
- `Congestion Window (CWND)`: How many bytes can be sent without overflowing the routers?

From those two windows follows the `Sender Window`. determined by $\min(\mathrm{CWND}, \mathrm{RWND})$.

Congestion Detection

There are essentially three ways to detect congestion:

- The network could tell the source but the signal itself could be lost on the way.
- We could measure the packet delay but the signal probably will be noisy.
- We could measure the packet loss. This is a fail-safe signal that TCP already has to detect.

If we detect duplicated ACKs, we have a `mild congestion signal` and if we have a timeout, we have a `severe congestion signal`.

Reacting to Congestion

TCP's approach is to gently increase when not congested and to rapidly decrease when congested.

We start with the `Slow Start Phase`, that is, we set the CWND to 1 and increase it by one for each ACK we receive. However, once we have a rough estimate of the bandwidth, we need a more gentle adjustment algorithm.

The two possible options are:

- `Multiplicative Increase or Decrease (MID)`, i.e., $\mathrm{CWND} = a \cdot \mathrm{CWND}$.
- `Additive Increase or Decrease`, i.e., $\mathrm{CWND} = b + \mathrm{CWND}$.

TCP implements additive increase multiplicative decrease (AIMD): After each ACK, we increment the CWND by $1/\mathrm{CWND}$. Per RTT, the increase is therefore at most 1. One might as the question on when the sender leaves the slow-start phase and starts AIMD. We introduce a `slows start threshold`, and adapt it as a function of congestion. On a timeout, we set $\mathrm{SSTHRESH} = CWND/2$.

3.3.8 QUIC - Quick UDP Internet Connections

`QUIC` is the biggest transport change in the last 30 years. It is a transport layer protocol designed by Google to improve performance of HTTPS. Broadly explained, QUIC takes everything from the TCP header and puts it into the data field of a UDP packet.

- QUIC rolls in TCP and TLS connection negotiation. Therefore, a QUIC handshake is enough (instead of a separate TCP and a separate TLS handshake).
- Based on a cookie, QUIC allows for 0-RTT session resumption where a *secret* learned from a previous session is used to encrypt/decrypt messages.

3.4 Sockets: the application and the transport interface

> A `socket` is a software abstraction by which an application process exchanges network messages with the transport layer in the operating system.

Note: Socket are an OS abstraction and ports are a networking abstraction. In this context, they do not correspond to any physical port like on routers and switches, but are rather logical interfaces that one uses.

3.4.1 Ports

To determine which app (*socket*) gets which packets, `ports` act as a 16-bit transport layer identifier . A packet then carries source/destination port number is the transport header and the OS stores the mapping between sockets and ports.

The 16-bit address space is built up in the following way:

- Ports 0-1023 are *well known* and there is an agreement on which services run on these ports, e.g., SSh:22, HTTP:80, HTTPS:443, etc. A client using these services therefore knows the appropriate port and can listen on it directly.
- Ports 1024-65535 are *ephemeral* and ar given to clients ar random.

3.4.2 Multiplexing & Demultiplexing

A host receives IP datagrams with a source and destination IP address as well as a source an destination `port number`. The IP address and the port are then used to deliver the segment to the appropriate socket.

The mappings are as follows:

- For UDP (`SOCK_DGRAM`): OS stores (local port, local IP address) $\Leftrightarrow$ socket
- For TCP (`SOCK_STREAM`): OS stores (local port, local IP, remote port, remote IP) $\Leftrightarrow$ socket

This is due to the nature of the protocols. Since TCP is connection oriented, it needs to remember the source IP and port number of an incoming IP datagram, whereas UDP as a connectionless protocols need not remember that information.

4. Algorithms in Networking

4.1 Three example algorithmic problems in networking

4.1.1 Example 1: Traffic engineering

`Max-Flow-Problem` : Given a network, displayed as a directed graph, what is the maximum traffic from some endpoint S to another endpoint T ? Solved by determining the min-cut of the graph (which will be equal to the max-flow).

4.1.2 Example 2: Matchings and circuits

`Matchings` are often seen inside circuit switches that consist of micro-mirrors and we want to allow as many concurrent connections/circuits as possible.

4.1.3 Example 3: Shortest paths

Again, assume a network, shown as a directed graph, and trying to find a shortest path from some source node S to a destination node T.

4.2 Linear programming: a powerful, generic tool

4.2.1 What is a linear program?

A `linear program` is based on the following three definitions one needs to assign to define a linear program:

1. `Variables` : E.g., flows on edges, distance from start, etc. Must be real numbers, $x \in \mathbb{R}$
2. `Objective` : E.g., maximize flow, minimize distance, etc. Must be a linear combination of the variables.
3. `Constraints` : Flow on edge $\leq$ capacity of edge, etc. Must too be a linear combination of variables.