Chair of Distributed Systems and Security
School of Computation, Information and Technology
Technical University of Munich

Computer Networking and IT Security (INHN0012)

Tutorial 11

Problem 1 Flow and congestion control with TCP

The most widely used transport protocol on the Internet is TCP. It implements mechanisms for flow and congestion control.

a)* Discuss the differences between flow and congestion control. What are the objectives of each mechanism?

- **Flow control:**
  Prevention of overload situations at the receiver
- **Congestion control:**
  Adaption to overload situations in the network

b) Assign each of the following terms to TCP flow or congestion control:

- Slow-Start
- Receive window
- Congestion-Avoidance
- Multiplicative-Decrease

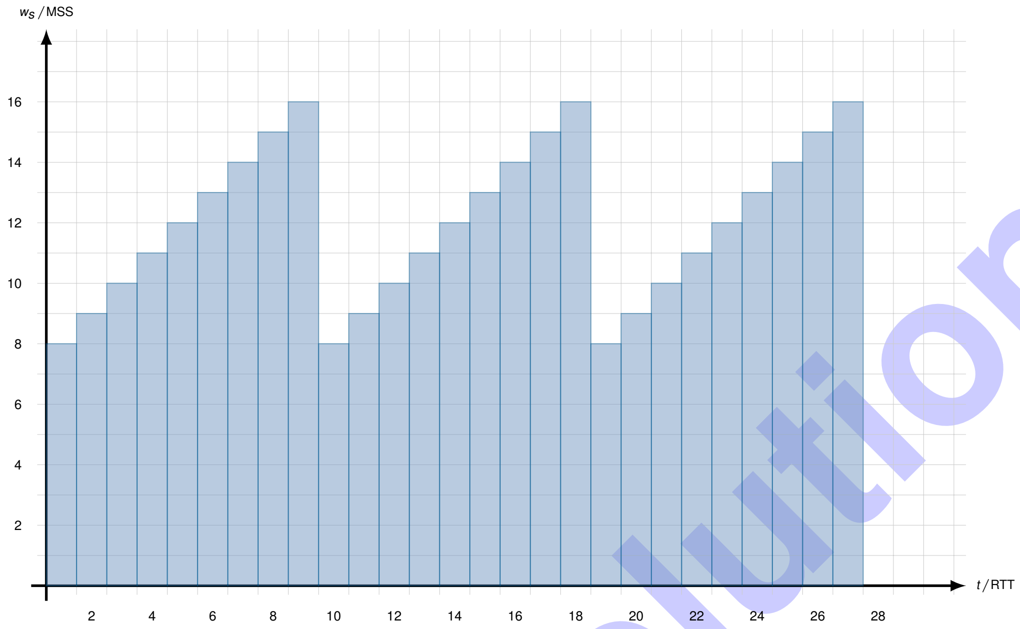
Only the receive window is part of flow control, since the receiver uses it to inform the sender of the maximum amount of data it can send at once.
All remaining terms belong to congestion control, with slow-start and congestion-avoidance being the two congestion control phases of a TCP connection. Multiplicative decrease, on the other hand, is the halving of the congestion control window when a segment is lost.

To analyze the data rate that can be achieved with TCP, we consider the course of a contiguous data transmission in which the slow-start phase has already been completed. TCP is therefore operating in the congestion-avoidance phase. We define the individual windows as follows:

- Send window W_s, $|W_s| = w_s$
- Receive window W_r, $|W_r| = w_r$
- Congestion-control window W_c, $|W_c| = w_c$

We assume that the receive window is arbitrarily large, so that the send window is determined solely by the congestion-control window, i. e., $W_s = W_c$. No losses occur as long as the send window is smaller than a maximum value x, i. e., $w_s < x$.
If a full send window is acknowledged, the currently used window increases by exactly 1 MSS. If the send window reaches the value x, exactly one of the sent TCP segments is lost. The sender detects the loss by receiving the same ACK number multiple times. The sender then halves the congestion-control window, but still remains in the congestion-avoidance phase, i. e., no new slow-start takes place. This approach corresponds to a simplified variant of TCP-Reno (cf. lecture).
As concrete numerical values, we assume that the maximum TCP segment size (MSS) is 1460 B and that the RTT is 200 ms. Let the serialization time of segments be negligible compared to the propagation delay. Segment loss occurs as soon as the send window reaches a size of $w_s \geq x = 16$ MSS.

c)* Create a graph plotting the current size of the send window w_s, measured in MSS, over the time axis t, measured in RTT. In your diagram, at time $t_0 = 0\,\text{s}$, the send window size has just been halved, so that $w_s = x/2$ holds. Draw the diagram for the time interval $t = \{0, \ldots, 27\}$.



d)* How much time elapses before the congestion control window is reduced again after a segment loss as a result of another segment loss?

After segment loss, w_c is reduced to $x/2$ and then increased again by 1 MSS per fully acknowledged window. Since the serialization time is negligible, a total of w_c segments can be sent at time t_0, which are acknowledged at time $t_0 + \text{RTT}$. Consequently, for the time until the maximum value is reached again, we get

$$T = \left(\frac{x}{2} + 1\right) \cdot \text{RTT} = 9 \cdot 200\,\text{ms} = 1.8\,\text{s}.$$

e)* Determine the average loss rate L.
Note: Since the behavior of TCP is periodic in this idealized model, it is sufficient to consider only a single period. Set the total number of transmitted segments in relation to the number of lost segments (specification as a truncated fraction is sufficient).

First, we determine the number n of segments transmitted during each „sawtooth“

$$
\begin{aligned}
n &= \sum_{i=x/2}^{x} i = \sum_{i=1}^{x} i - \sum_{i=1}^{x/2-1} i = \frac{x \cdot (x+1)}{2} - \frac{\left(\frac{x}{2}-1\right) \cdot \frac{x}{2}}{2} \\
&= \frac{x^2 + x}{2} - \frac{x^2}{8} + \frac{x}{4} \\
&= \frac{3}{8}x^2 + \frac{3}{4}x \\
&\overset{x=16}{=} 108
\end{aligned}
$$

Exactly one segment is lost per „sawtooth“. Thus, the loss rate is

$$
L = \frac{1}{\frac{3}{8}x^2 + \frac{3}{4}x} = \frac{1}{108} \approx 9.26 \cdot 10^{-3}
$$

f) Using the results from subtasks d) and e), determine the average achievable transmission rate in kB/s during the TCP transmission phase under consideration.
Note: Use the exact value (fraction) from subtask e).

The resulting data rate is

$$
\begin{aligned}
r_{TCP} &= \frac{n \cdot \mathrm{MSS}}{T} \cdot (1 - \theta) \\
&= \frac{108 \cdot 1460\,\mathrm{B}}{1.8\,\mathrm{s}} \cdot \frac{107}{108} \\
&= \frac{107 \cdot 1460\,\mathrm{B}}{1.8\,\mathrm{s}} \\
&= \frac{1562200}{18}\,\mathrm{B/s} \\
&\approx \frac{1562}{18}\,\mathrm{kB/s} \approx 86.79\,\mathrm{kB/s}.
\end{aligned}
$$

g)* What is the maximum transmission rate that could be achieved over the channel using UDP without causing congestion? Take into account that the UDP header is 12 B shorter than the TCP header without options.

Apparently, 15 MSS can be transmitted reliably. In addition, a UDP datagram carries 12 B more payload data than a TCP segment.Thus, we obtain

$$
\begin{aligned}
r_{UDP} &= \frac{15 \cdot (\mathrm{MSS} + 12\,\mathrm{B})}{\mathrm{RTT}} \\
&= \frac{15 \cdot (1460\,\mathrm{B} + 12\,\mathrm{B})}{0.2\,\mathrm{s}} \\
&= \frac{15 \cdot 1472\,\mathrm{B}}{0.2\,\mathrm{s}} \\
&= 110.40\,\mathrm{kB/s}.
\end{aligned}
$$

Problem 2 Compression: Huffman Coding

Given the alphabet $\mathcal{A} = \{a, b, c, d\}$ and the message

$$m = \text{aabcbdacababbbbcbddbbbaababdbdbb} \in \mathcal{A}^{32}.$$

a)* Determine the occurrence probabilities p_i of each character $i \in \mathcal{A}$ in the message m.

From the character frequencies it follows:

$$p_a = \frac{8}{32} = \frac{1}{4}, \ p_b = \frac{16}{32} = \frac{1}{2}, \ p_c = \frac{3}{32} \approx 0.09, \ p_d = \frac{5}{32} \approx 0.16$$

b) Determine the information content $I(i)$ of each character from $\mathcal{A}$.

For the information content we get:

$$\begin{aligned} I(a) &= -\log_2(p_a) = 2\,\text{bit} \\ I(b) &= -\log_2(p_b) = 1\,\text{bit} \\ I(c) &= -\log_2(p_c) \approx 3.42\,\text{bit} \\ I(d) &= -\log_2(p_d) \approx 2.68\,\text{bit} \end{aligned}$$

c) The message m originates from a message source X. Based on the previous results, determine the source entropy $H(X)$.

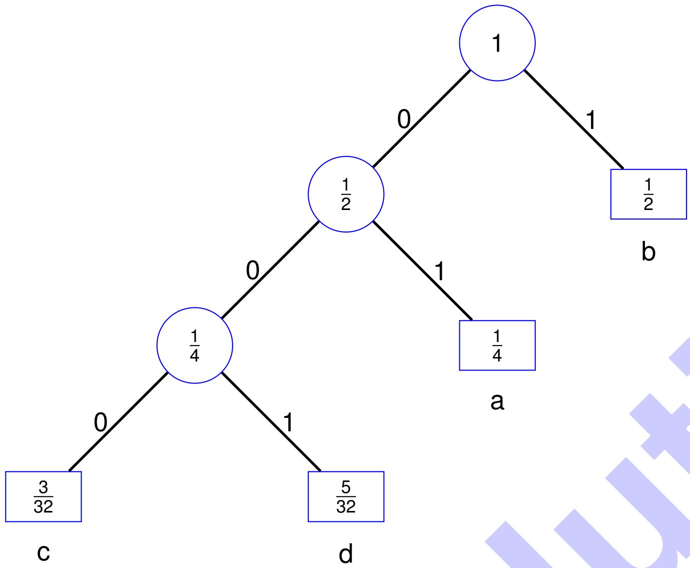
The source entropy is nothing more than the sum of the information content of the individual characters weighted with the occurrence probabilities:

$$H(X) = \sum_{i \in \mathcal{A}} p_i I(p_i) \approx 1.74\,\text{bit}$$

This means that the characters of the source X can be encoded with an average of 1.74 bit per character.

d) Now determine a binary Huffman code C for this message source.

See lecture slides. Starting with the two characters with the lowest probability of occurrence, a tree is constructed starting with the leaves (the characters). In each step, the two nodes or leaves are always combined so that the sum of the occurrence probabilities over all nodes or leaves is minimal:



The edges are labeled with 0 or 1 arbitrarily. The code can now be easily read by starting from the root and reading the edge labels: $C = \{a \mapsto \texttt{01}, b \mapsto \texttt{1}, c \mapsto \texttt{000}, d \mapsto \texttt{001}\}$

Characters with high occurrence probabilities are given short codewords. Also, it is easy to verify that C is prefix-free: no codeword is a prefix of any other codeword. The characters are defined only at the leaves of the tree, but not at the inner nodes. This facilitates decoding.

e) Determine the average codeword length of C.

The average codeword length is the sum of the codeword lengths weighted by the occurrence probabilities. Let $l(c)$ be the length of a codeword in C and $c(i)$ be the function that maps a character $i \in \mathcal{A}$ to a codeword from C. Then we get:

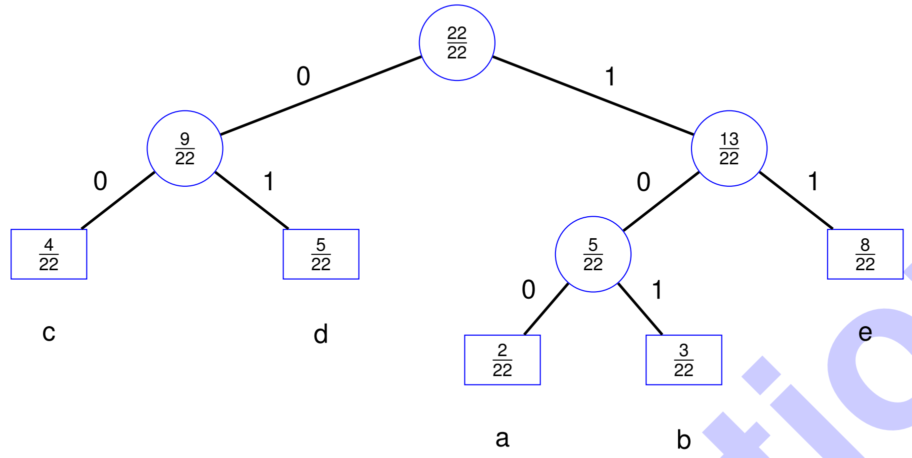
$$\bar{l}_C = \sum_{i \in \mathcal{A}} p_i \cdot l(c(i)) = 1.75\,\text{bit}$$

f) Compare the average codeword length of C with the codeword length of a uniform[1] binary code.

The shortest uniform code has an average codeword length of $\bar{l}_U = 2\,\text{bit}$. So the saving is about 12.5 %.

[1] A code is called *uniform* if all codewords have the same length

g) **Homework:** Create a binary Huffman-Code C' for another message source Q', which emits characters of the alphabet $\chi = \{a, b, c, d, e\}$.
The relative probabilities of the characters are $p_a = \frac{2}{22}, p_b = \frac{3}{22}, p_c = \frac{4}{22}, p_d = \frac{5}{22}, p_e = \frac{8}{22}$.



$C' = \{a \mapsto \texttt{100}, b \mapsto \texttt{101}, c \mapsto \texttt{00}, d \mapsto \texttt{01}, e \mapsto \texttt{11}\}$
In general, there can be multiple correct solutions for the tree. In this particular case, it is possible to combine (c) with either (d) or the subtree (a,b). However, regardless of the chosen variant, the lengths of the respective codewords remain the same.