Technische Universität München
Lehrstuhl Informatik VIII
Prof. Dr.-Ing. Georg Carle
Christian Grothoff, Ph.D.
Stephan M. Günther, M.Sc.

Master Course Computer Networks

Homework 3
(submission until December 3rd into INBOX located in front of 03.05.052)

Note: Subproblems marked by * can be solved without preceding results.

SYN Cookies

The lecture introduced a DoS attack referred to as SYN flooding. Without additional precautions, TCP is especially susceptible to this form of attack. In this exercise you will investigate a countermeasure against SYN flooding for TCP, called *SYN Cookies*. A reference to start with for this homework is RFC 4987.

a)* Briefly describe the problem of SYN flooding and argue whether UDP can be affected as well.

When performing a SYN-flooding attack, a client (or group of clients) send SYN-packets to a server, accept the returning SYN/ACK but to not complete the three-way handshake by sending the final ACK. Instead, the client silently discards the SYN/ACK and starts a new handshake.
However, the server cannot know whether the final packet of the initial handshake is delayed or will never arrive. Thus, it has to keep state (the socket in half-open state) until some timeout expires.

If a server is flooded with thousands of connection attempts within a short time period, it may exceed its maximum number of concurrent connections and thus become unresponsive.

Also see RFC 4987 Section 2.2.

b)* Describe the concept of TCP SYN cookies in your own words. Sketching the TCP handshake and explaining the values being exchanged might be helpful.

See RFC 4987 Section 3.6.

c)* Have a look at RFC 2960 and sketch the SCTP handshake and describe the cookie mechanism used by SCTP.

See RFC 2960 Section 5, in particular Section 5.1.3 – 5.1.6.

TCP Congestion Avoidance

In contrast to UDP, TCP offers a congestion avoidance algorithm that tries to dynamically adapt the sender's rate to the available capacity on the link. Furthermore, TCP allows a receiver to throttle the sender's rate if necessary. Read RFC 2581 and answer the following short questions.

a)* What is the difference between congestion control and flow control?

Congestion control tries to avoid congestion along the path from sender to receiver, i. e., controls

overload within the network. Flow control allows the receiver to throttle the sender when necessary, i. e., controls overload at the receiver.

b)* Sketch a typical development of the TCP sender window for both variants Tahoe and Reno over time, starting at the time when the TCP connection is established. Mark and name the different phases.

See Figure 1: The main difference between both TCP versions is that Tahoe treats three duplicate ACKs like a timeout, i. e., a slow start is performed up to the actual slow start threshold (which is half of the congestion window when the duplicate ACKs occurred).
In contrast, Reno performs a *fast recovery* when three duplicate ACKs are detected, i. e., it starts with a congestion avoidance phase at the new slow start threshold. Only in case of a timeout event, Reno starts over with a new slow start.

Now assume a TCP connection over a satellite link. The average RTT is 800 ms and the link's available bandwidth is 24 Mbit/s. Assume that there is no packet loss before the link's bandwidth is achieved by the TCP connection.

c) Estimate the minimum amount of time necessary until an ordinary TCP connections fully utilizes the available bandwidth.

Full utilization means that the sender can continuously transmit at 24 Mbit/s for at least one RTT without receiving an acknowledgement. Thus, the congestion window must be

$$c_{\text{win}} \geq 24\,\text{Mbit/s} \cdot 0.8\,\text{ms} = 2.4\,\text{MB}.$$

The very first problem is that, by default, TCP's maximum window size is limited by the 16 bit value in the header. The maximum congestion window is thus $c_{\text{win,max}} = 65535\,\text{Byte}$. As a result, the congestion window will never reach the necessary size.

d) Describe how TCP window scaling can help to mitigate the problem.

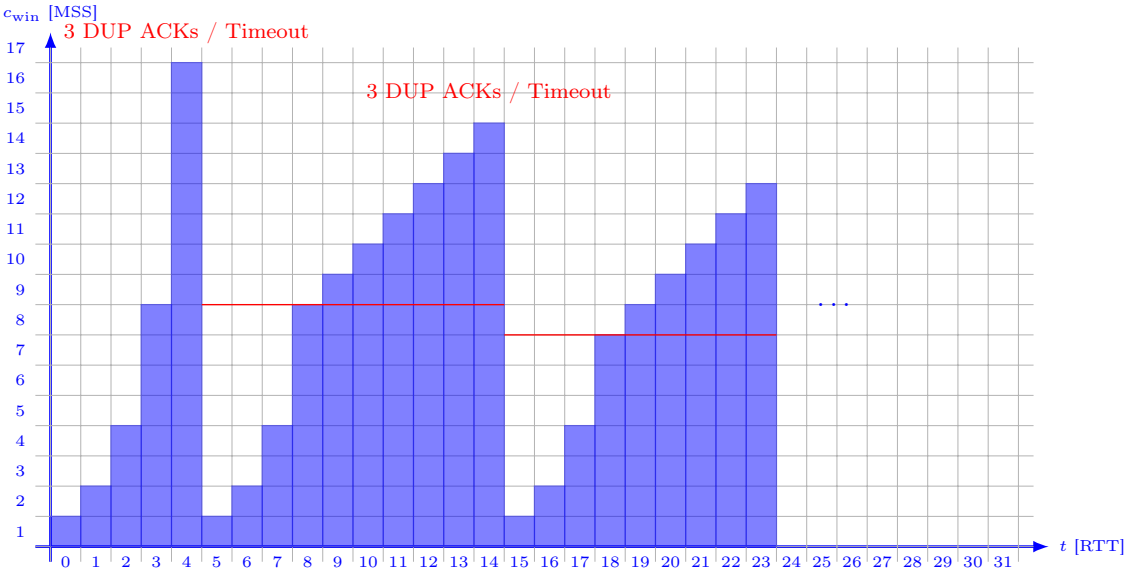
Also see RFC 1323: Window scaling is a TCP option which allows to interpret the window in multiples of 2^x Byte. This way the window can be scaled such that it can indeed receive the minimum required size of 2.4 MB. The value x can be calculated as follows:

$$\begin{aligned}\left(2^{16}-1\right)\cdot 2^x &\geq 2.4\cdot 10^6\\ x &\geq \log_2\left(\frac{2.4\cdot 10^6}{2^{16}-1}\right) \approx 5.19\\ \Rightarrow x &= 6\end{aligned}$$

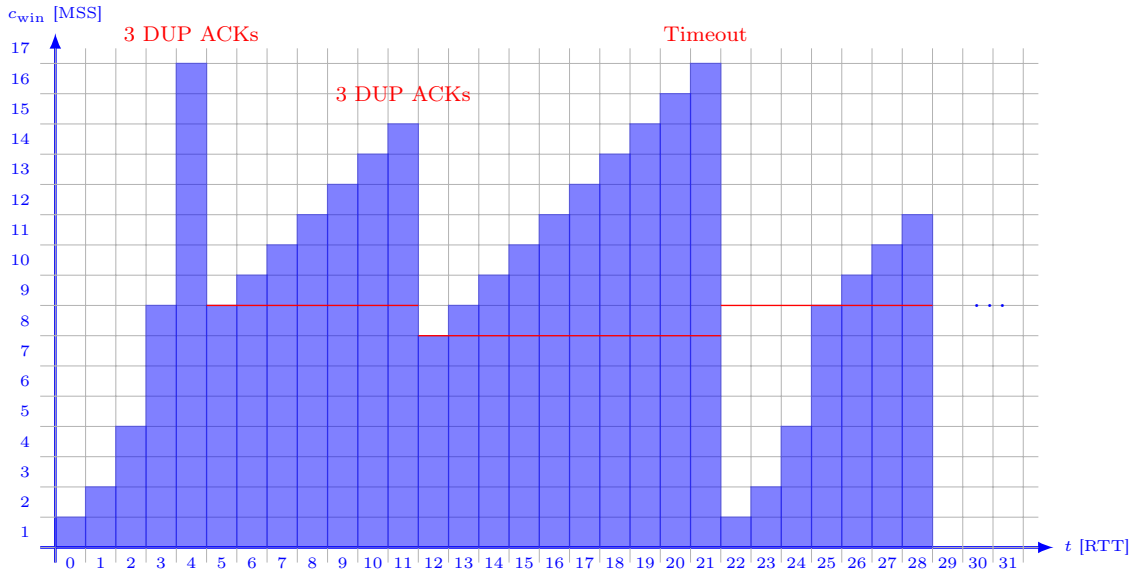
However, there remains a problem: Although the congestion window now may reach the desired size, it still grows slowly because its increments are *not* affected by the scaling. This means, during congestion avoidance it still grows by 1 mss per RTT only.
Assuming that the window was just cut in half due to a lost segment, i. e., $c_{\text{win}} = 1.2\,\text{MB}$, and given MSS = 1460 Byte (1500 Byte − IPHeader − TCPHeader), we obtain

$$\Delta t = \frac{1.2\,\text{MB}}{1460\,\text{Byte}} \cdot 0.8\,\text{s} \approx 658\,\text{s}$$

until the window is fully utilized again. Consequently, window scaling just enlarges the maximum window but does not reduce the time until some window size is reached – except for the case the desired window size is unreachable without scaling.



(a) TCP Tahoe



(b) TCP Reno

Figure 1: Congestion window of TCP Tahoe (a) and TCP Reno (b). The red line indicates the slow start threshold.