March 17 Lecture Transcript – (TCP Timers)

Slide 1:

During the last lecture, we started the discussion about 4 important timers provisioned by TCP protocol and implemented by the TCP module/software of every operating system/computer. Two of these timers were discussed last time …

The 3rd TCP timer (so-called Persistence Timer) – which is discussed in this slide – is intended to deal with a situation involving the following sequence of events:

1) the receiving machine (in an ongoing TCP connection) has closed its ReceiveWindow to 0;
2) then, at some point in time, the ReceiveWindow opens (i.e., its size increases) and the receiving machine sends a packet-update about this change to the sending machine;
3) however, this packet-update gets lost during transmission …

Clearly, due to 3, the sending machine would continue 'halting' the transmission of its packets, even though there are/were available buffer resources at the receiving machine.

In order to prevent the above described 'transmission deadlock' from happening, the sending machine initiates/starts a TCP 'persistence timer' ever time it receives a 0 ReceiveWindow notification from the receiving machine. Once 'persistence timer' reaches some predefined maximum value (i.e., longest wait time), the sending machine will send a 1-byte packet probe to the receiving machine in hope that this packet will trigger a positive response - i.e., it will cause the receiving machine to send an acknowledgment packet carrying a ReceiveWindow value greater than 0.

Now, it should be noted that

- The max value of persistence timer is initially typically set to 2x the value of 'roundtrip transmission time' (2RTT). (See slide 3 for the actual definition of RTT.)
- Once this amount of time (2RTT) elapses, the first 1-byte probe packet is sent to the receiving machine, and the maximum value of persistence time is increased to 2 x the initial value of maximum persistence timer (i.e., 4RTT).
- If there is no response to the first 1-byte packet probe within 4RTT, the second 1-byte probe is sent, and the maximum value of persistence time is increased to 2 x the previous value of maximum persistence timer (i.e., 8RTT). Etc.
- This pattern of double-increase of the maximum persistence timer is continued until the maximum persistence time reaches 60 seconds, after which point 1-byte probes are sent in regular intervals – one 1-byte probe every 60 seconds …

Slide 2:

This slides provides a visual illustration of what has been described in Slide 1.

Slide 3:

During the previous lecture, we discussed the importance of the so-called **Retransmission Timeout** timers. Recall, for every transmitted TCP packet (P), the sending machine initiates one (i.e., its respective) Retransmission Timeout (RTO) timer, and if the acknowledgment for P does not arrive within RTO amount

of time – either because P itself or its acknowledgment have been lost - the given packet (P) is retransmitted.

Now, clearly, deploying a fixed/predefined value for RTO would not be wise, since:

- For different TCP connections the amount of time that elapses between the transmission of a packet and the reception of its respective acknowledgment are different. (Btw, this time is generally referred to as **Roundtrip Transmission Time** – RTT, and is illustrated in the central figure in this slide.) For example, the RTT for a TCP connection between two computers sitting on the same LAN would be very different form the RTT for a TCP connection between two computers located in different LANs on different continents.
- Moreover, even if we consider one single TCP connection between the same 2 computers, the value of RTT for this connection could change over time – RTT will be shorter if there is no congestion along the transmission path between the 2 computers, and RTT will get longer as the congestion along the transmission paths occurs/increases.

In most real-world implementations of the TCP protocol, every TCP connection calculates/sets its own RTO usually to the value of $2xRTT_{current}$, where $RTT_{current}$ is the most recent real-time measurement/evaluation of RTT for the given connection. Now, the ways of how $RTT_{current}$ can be actually evaluated is discussed on the subsequent slide.

Slide 4:

Probably the simplest a way of estimating $RTT_{current}$ for a given TCP connection would be to measure the value of RTT for every transmitted packet (e.g., if there are K packet transmitted, there will be K RTT values measured), and then find the (simple) average over all the measured RTTs. The mathematical formula for finding this average is provided in the upper half of the slide.
However, although simple this way of estimating $RTT_{current}$ would not be very useful/accurate, as it would give the same weight/importance to the old RTT measurement as to the ones that are more/most recent.

Another way of estimating $RTT_{current}$ would be to use the so-called Exponential / Smoothed Average formula, as shown in the lower part of the slide. In the given formula, the value of $RTT_{current}$ for time (K+1) (marked as SRTT(K+1)) is calculated as a weighted sum of the following two:
- the most recent actual measurement of RTT at time (K+1), and
- the previous estimate of $RTT_{current}$ as it was obtained at time K (SRTT(K)).

In the provided formula, parameter α is the so-called weighting factor, and it can be adjusted to give more/less weight to the old vs. most recent measurement(s) of RTT. Clearly, for very small α, the importance of old RTT measurements will be small, while the value of $RTT_{current}$ will be mostly determined/influenced by the value of the most recent RTT measurement. On the other hand, large values of α will have an opposite effect.

Slide 5:

The graph provided in this slide shows the effect of α on the value of estimated $RTT_{current}$ (i.e., the value of SRTT from the previous formula), given a set of observed/measured RTT values. It should be clear from the graph that in this particular case, where t observed RTT keep changing/increasing over time …

- the value of α = 0.875 results in a rather poor (under)estimation of $RTT_{current,}$, as more weight is given to old(er) and thus smaller values of RTT;

- α = 0.5, on the other hand, results in a much better estimate/tracking of RTT values, as it strikes a balance between the old and the new RTT measurements.

Slide 6:

One problem in the estimation of $RTT_{current}$ arises in the case of TCP connection(s) with packet loss and or delay that results in packet retransmission.

Namely, imagine a scenario illustrated in the figure provided on this slide:

- A packet (P) is sent from machine A to B.
- The acknowledgment for this packet is delayed (beyond the usual RTO time), so as a result packet P is resent.
- However, soon after P gets resent, the original/delayed acknowledgment for P arrives. Just by looking/examining this acknowledgment, machine A would have no way of knowing whether the received acknowledgment corresponds to the originally sent P or the resent P. As a result, machine A would not be in the position to accurately estimate RTT for this packet …

In order to avoid the above described problems/challenges, the so-called Karn's Algorithm proposes that the estimation of $RTT_{current}$ does not rely on (i.e., does not take into consideration) retransmitted packets and their respective RTT values ...