Praktikum Simulative Untersuchung von Protokollfunktionen des Internet

Aufgabenblatt 12 [1]

Vorbereitung

Aufgabe 1: Handsimulation von Congestion Control
Lesen Sie sich die Aufgabenstellung im praktischen Teil gut durch. Füllen Sie dann für die angegebenen Ereignisse die folgende Tabelle aus:

Ereignis	`lastAckSeqNo`	`nextSendSeqNo`	`congWindow`	`ssthresh`
–	10	16	10	128
send DATA 16-19				
receive ACK 11-20				
send DATA 20-39				
receive ACK 21-26				
send DATA 40-51				
TIMEOUT				
send DATA 26				
receive ACK 27				
send DATA 27,28				
receive ACK 28,29				
send DATA 29-32				
receive ACK 30-33				
send DATA 33-40				
receive ACK 34-39				
send DATA 41-52				
receive ACK 40-47				
send DATA 53-60				
receive ACK 48-53				
send DATA 61-67				
receive ACK 54-60				
send DATA 68-74				
receive ACK 61-68				

Praktischer Teil

Denken Sie bitte –wie immer bei den praktischen Aufgaben– an aussagekräftige Debug-Ausgaben. Die Sourcen für diese Aufgabe befinden sich im Verzeichnis `protsim12`.

Im letzten Termin haben Sie gelernt, wie sich schnelle Sender mittels Flusskontrolle an langsame Empfänger anpassen können. Was passiert jedoch, wenn viele Applikationen im Netz senden und dadurch Teile des Netzes überlasten? Die Abwehr solcher Überlastsituationen ist die Aufgabe der Staukontrolle (Congestion Control). Die Variante, die Sie hier kennen lernen, orientiert sich im wesentlichen an der Congestion Control von TCP mit dem wesentlichen Unterschied, dass bei TCP byte-weise numeriert wird, während wir zur Vereinfachung Paket-orientiert arbeiten.

Das Congestion Window (`congWindow`) gibt an, wieviele unbestätigte Pakete aus Sicht der Staukontrolle maximal unterwegs sein dürfen. Da gleichzeitig auch Flusskontrolle stattfinden soll, dürfen jedoch höchstens das Minimum aus Congestion Window und Receiver Window gesendet werden.

[1] Stand: 12. Oktober 2023

Unsere Staukontrolle arbeitet wie in der Vorlesung beschrieben in zwei Phasen, der Slow-Start-Phase und der Congestion-Avoidance Phase. Die Variable `ssthresh` gibt an, bis zu welchem Wert von `congWindow` (einschließlich) der Slow-Start-Algorithmus verwendet werden soll. Die Congestion-Avoidance Phase beginnt also erst bei `ssthresh+1`.

Aufgabe 2: Staukontroll-Implementierung
Implementieren Sie nun die Staukontrolle, indem Sie ihre Datei `transport/TransportEndpointData.cc` aus der letzten Aufgabe kopieren und erweitern. Übernehmen Sie auch `transport/TransportEndpoint.cc`.

Beachten Sie bitte folgende Hinweise:

- Congestion Control soll nur stattfinden, wenn der Parameter `congestionControl` wahr ist.
- Erweitern Sie in `handleData()` die Behandlung von Acknowledgements so, dass
  - in der Slow-Start-Phase das Congestion Window `congWindow` für jedes empfangene Acknowledgement-Paket um eins erhöht wird,
  - in der Congestion-Avoidance-Phase das Congestion Window um eins erhöht wird, wenn eine der aktuellen Größe des Congestion Windows entsprechende Anzahl von Acknowledgement-Paketen eingegangen ist. Benutzen Sie die Variable `congAvoidCounter` um das festzustellen (am einfachsten ist herunterzählen auf 0).

  Beachten Sie bitte, dass es hierbei um die Anzahl empfangener Acknowledgement-Pakete geht, nicht darum wieviele Datenpakete bestätigt wurden.
- Passen Sie `handleDataTimer()` so an, dass bei jedem Timeout das Congestion Window auf eins reduziert wird. `ssthresh` soll auf die Hälfte des vorherigen Congestion Window gesetzt werden, mindestens jedoch auf 2. Setzen Sie ggf. auch `congAvoidCounter` auf einen sinnvollen Wert.
- Passen Sie `doSending()` so an, dass bei eingeschalteter Congestion Control das aktuelle Congestion Window beim Senden beachtet wird.
- Zeichnen Sie die aktuelle Größe des Congestion Windows nach jeder Änderung im Vektor `congWindowVector` auf.
- Zeichnen Sie die aktuelle Größe des Slow-Start-Thresholds vor und nach jeder Änderung im Vektor `ssthreshVector` auf.

Aufgabe 3: Testen
Zum Debuggen können Sie auch wieder das `protsim_tcp_verify.pl`-Skript benutzen. Dazu müssen Sie zuvor alle Subvektoren des Runs per `splitpsvec.py` extrahieren.

Führen Sie danach zum Testen die Runs 1-5 (ohne Congestion Control) und 6-10 (mit Congestion Control) aus (mit `./runAll`). Betrachten Sie von einigen Applikationen die Vektoren für `ssthresh` und `congWindow` und überlegen Sie, ob die Ergebnisse plausibel sind. Splitten Sie dazu den Outputvektor `transpC06.vec` mit `splitvec.py` in alle Subvektoren, betrachten Sie anschließend mit dem `plot` Script den conWinows- und ssthreshVektor der Applikation 1 des linken Knotens.

Vergleichen Sie mit `ps_eval_arq.py` für die Runs 1-5 bzw. 6-10, welchen Einfluss die Congestion Control auf den Goodput hat.