Rechnernetze und verteilte Systeme
Theorie 5: Transportschicht

Fachgruppe Telekommunikationsnetze (TKN)

19. Februar 2024

Einleitung

Die folgenden Aufgaben werden gemeinsam im Tutorium bearbeitet. In der Veranstaltung Rechnernetze wird die SI-Notation verwendet. Beispiele für Präfixe: m = 10^{-3}, k = 10^3, M = 10^6, ki = 2^{10}, Mi = 2^{20}. „B“ bezeichnet Bytes, „bit“ Bits.

Übung 1 *TCP Basics*

Sie haben in der Vorlesung die Funktionsweise von TCP kennengelernt. Beantworten Sie in diesem Kontext die folgenden Fragen:

1. Welche Pakete werden beim TCP-Handshake ausgetauscht?
2. Welche Pakete werden ausgetauscht, wenn der eigene Rechner die TCP-Verbindung abbauen möchte? Was passiert, wenn der entfernte Rechner noch ausstehende Daten hat?
3. Das Internet gilt als sog. Best-Effort-Netzwerk. Konkret heißt das, das Pakete verfälscht werden können, komplett verloren gehen oder die Sendereihenfolge nicht erhalten wird. Überlegen Sie sich wie diese Effekte zustande kommen können!
4. Welche Mechanismen nutzt TCP um die Probleme aus c) zu lösen?

Lösung

1. SYN, SYN-ACK, ACK
2. FIN, FIN-ACK, ACK
3.
   - Biterror können (insbesondere bei drahtloser) Übertragung entstehen.
   - Paketverluste entstehen oft bei Überlastung im Netzwerk oder wenn untere Schichten fehlerhafte Pakete verwerfen.
   - Nicht jedes Paket nimmt zwangsläufig die gleiche Route zum Ziel. Daher kann ein späteres Paket durch einen kürzeren Weg vorhergehende

Pakete überholen.

4.
   - Biterror: Checksummenberechnung und Retransmisson (nach NACK oder Timeout),
   - Paketverlust: Sequenznummer, Puffer beim Retransmission nach Timeout bei fehlendem ACK
   - Reihenfolge: Sequenznummern, ggf. Anforderung von fehlenden Paketen durch Receiver.

Übung 2 *ARQ*

Sie haben in der Vorlesung verschiedene Fehlerkontrollmechanismen (ARQ) kennengelernt. Bearbeiten Sie in diesem Kontext folgende Aufgaben:

1. Wie erkennt ein ARQ-Mechanismus überhaupt Fehler?
2. Was ist ein kumulatives ACK?
3. Wie funktioniert das GoBackN-Verfahren? Erklären Sie es in eigenen Worten Ihren Komilliton:innen!
4. Vergleichen Sie die vorgestellten ARQ-Mechanismen anhand ihrer Komplexität sowohl für Sender als auch Empfänger!
5. Angenommen es wird TCP genutzt und der Empfänger hat gerade Byte 4711 empfangen. Welche Sequenznummer schreibt der Empfänger in sein ACK-Paket?

Lösung

1. Mittels der Kombination von Ackknowledgements und Timeouts.
2. Bei einem kumulativen ACK bestätigt ein ACK implizit auch alle vorangegangenen Pakete.
3. Das GoBackN-Verfahren erlaubt dem Sender mehrere Pakete zu senden, bevor ein ACK erhalten wurde. Die maximale Anzahl der Pakete ist durch die Fenstergröße beschränkt. Beim Senden des ersten Pakets, setzte der Sender einen Timer und puffert alle gesendeten unbestätigten Pakete. Wenn der Timer abgelaufen ist, werden alle unbestätigten Pakete erneut gesendet. Der Empfänger antwortet mit kumulativen ACKs. Die SQN in dem ACK bestätigt den erfolgreichen Empfang aller vorherigen Pakete bis einschließlich der SQN. Der Empfänger akzeptiert Pakete also nur in der richtigen Reihenfolge und benötigt keinen Puffer.

4. **Für alle ARQ-Verfahren gilt**

   - Sowohl Sender als auch Empfänger müssen die Sequenznummern speichern.
   - Der Sender muss Timeouts setzen.

   Send-and-Wait

   - Sender muss 1 (das aktuelle) Segment puffern bis es bestätigt wird.
   - Empfänger braucht nur einen Empfangspuffer von 1 Segment.

   GoBackN

   - Sender muss das gesamte Fenster puffern
   - Der Empfänger muss sich nur die Sequenznummer des zuletzt akzeptierten Pakets merken.
   - Der Empfänger akzeptiert die Pakete nur in der richtigen Reihenfolge.

   Selective-Repeat

   - Sender und Empfänger müssen jeweils 1 Fensterlänge puffern.
   - Empfänger muss sich merken welche Segmente noch fehlen.

Übung 3 *TCP Window*

TCP benutzt einen sog. Schiebefenstermechanismus, nicht nur zur Fehlerkontrolle sondern auch zur Fluss- und Überlast/Staukontrolle.

1. Was ist der Unterschied zwischen Flow- und Congestion-Control?
2. Wie wird der Timeout in TCP gewählt? Ist dieser konstant?
3. Was sind die Auswirkungen einer zu großen, bzw. einer zu kleinen Schätzung der RTT?
4. Wie interpretiert TCP Paketverluste? Geben Sie ein Beispiel, wann diese Interpretation falsch ist.

Lösung

1. **Flow Control** Kontrollmechanismus zur Verhinderung einer Überlastung

des Empfängers. Betrachtet nur die beiden Endpunkt der Kommunikation.

Congestion Control Kontrollmechanismus zur Vermeidung einer Überlastung einzelner Knotenpunkte im Netzwerk. Muss immer immer für das ganze Netz betrachtet werden.

2. Nein, der Timeout darf nicht konstant sein, da sich bspw. die Route der Verbindung jederzeit ändern kann. Bestimmung des Timeouts:
   - siehe VL Transportschicht Folie 131.
   - RTT wird kontinuierlich gemessen (Zeit von Segment gesendet bis zum ACK)
   - Der Timeout wird bestimmt aus einem gewichteten Durchschnitt der RTT-Messungen und einer gewichteten mittleren Abweichung

$$\begin{aligned} \text{EstimatedRTT} &= (1-\alpha)\cdot\text{EstimatedRTT} - \alpha\cdot\text{SampleRTT} \\ \text{DevRTT} &= (1-\beta)\cdot\text{DevRTT} + \beta\cdot|\text{SampleRTT} - \text{EstimatedRTT}| \\ \text{TimeoutInterval} &= \text{EstimatedRTT} + 4\cdot\text{DevRTT} \end{aligned}$$

   Je kleiner α und β sind, desto größer ist der Einfluss der vergangenen Messungen. Für große α und β lässt den Timeout aggressiver auf Schwankungen reagieren. Durch den DevRTT-Term wird der Timeout leicht höher gesetzt als der eigentliche RTT um sich gegen die Schwankungen abzusichern.

3. - Zu groß: Verlust von Segmenten wird zu verspätet erkannt.
   - Zu klein: Segmente werden fälschlicherweise wiederholt gesendet. Der Timeout für ACK wird aus der geschätzten RTT abgeleitet, ist diese zu gering läuft der Timeout (kurz) bevor die Pakete, bzw. deren Acknoledgements empfangen werden.

   Was die „richtige“ Schätzung ist, lässt sich im Allgemeinen nicht eindeutig definieren. In der Praxis verändert sich die RTT im Laufe der Verbindung. Die Schätzung bezieht daher die Kosten (im Sinne von investierten Resourcen, oder höherer Latenz) für falsche Schätzungen mit ein und versucht ein Optimum zu finden.

4. TCP interpretiert Paketverluste immer als Netzwerküberlast. Das setzt aber voraus, dass auf darunterliegenden Schichten keine Pakete verloren gehen, was bspw. bei Funkübertragungen nicht immer gewährleistet ist.

Übung 4 *GoBackN*

Nutzen Sie das verlinkte Onlinetool[1] um eine Simulationen von GoBackN durchzuführen. Nehmen Sie eine Fenstergröße von 5 Paketen an und spielen Sie verschiedene Szenarien durch (z.B. Verlust eines oder mehrerer Pakete und/oder Acknowledgements)

Interessant dürften extreme Fälle sein, z.B.:

- Das erste Paket kommt nicht an, aber die restlichen 4
- Alle 5 Pakete kommen an. Die ersten 4 ACKs gehen verloren, nur das 5. kommt an.

Hinweis

Das Onlinetool setzt den Timer komplett zurück für jedes ACK das ankommt. Im Tanenbaum und in der Musterlösung ist das nicht der Fall. Dort wird der Timer auf den Timeout des ältesten Pakets gesetzt. Ersteres ist zwar effizienter, z.B. falls das erste ACK verloren geht, aber letzteres hat höhere Effizienz wenn das letzte Paket im Fenster verloren geht, da es schneller wiederholt wird.

Übung 5 *Bit Flips*

Ein Paket der Länge n soll über einen fehleranfälligen Kanal übertragen werden. Die Bitfehlerwahrscheinlichkeit betrage p_b.

1. Wie groß ist die Wahrscheinlichkeit, dass das Paket ohne Fehler übertragen wird?
2. Wie groß ist der Erwartungswert der Anzahl der Versuche die unternommen werden müssen, bis das Paket einmal ohne Fehler übertragen wurde?
3. Welche Annahme liegt Ihrer Berechnung zugrunde und weshalb ist sie realistisch, bzw. unrealistisch?

Lösung

1. Ein Paket wird korrekt übertragen, wenn alle Bits korrekt übertragen wurden. Angenommen Bitfehler sind unabhängig voneinander ist die Wahrscheinlichkeit p_s dafür also $(1 - p_b)^n$. Für eine Paket mit $n = 1500bit$ und eine Bitfehlerwahrscheinlichkeit von $p_b = 10^{-7}$ wäre $p_s = 0.999$, die Fehlerwahrscheinlichkeit also 10^{-4}
2. Die Anzahl an Versuchen die bei einem Binomialexperiment bis zum Erfolg benötigt werden sind durch die geometrische Verteilung[a] beschrieben. Der Erwartungswert dieser Verteilung ist durch die Wahrscheinlichkeit der Binomialexperimente bestimmt, also hier p_s. Der Erwartungswert ist ent-

[1] https://www.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/

sprechend $E(X) = \frac{1}{p_s}$.

3. Die grundlegende Annahme dieser Berechnung ist, dass die Verteilung der Bitfehler unabhängig voneinander sind. Diese Annahme ist so nicht realistisch: Bitfehler treten häufig in Folge auf, sie sind "bursty". Ein Bitfehler (B) ist also wahrscheinlicher, wenn das vorherige Bit (A) schon fehlerhaft war, entsprechend sind diese Ereignisse nicht unabhängig voneiander: $P(A \cap B) \neq P(A) \cdot P(B)$.

   Solche vereinfachenden Annahmen sind dennoch legitim, da sie Berechnungen erleichtern (und gegebenenfalls überhaupt erst berechnbar machen), solange man sich der vereinfachenden Annahmen und deren Konsequenzen bewusst ist (und diese klar kommuniziert). In diesem Fall überschätzt die obige Berechnung die *Paket*fehlerwahrscheinlichkeit, da, ausgehend von einer gemittelten Gesamtfehlerwahrscheinlichkeit, ein Bitfehler lokal eine höhere Fehlerwahrscheinlichkeit bedeutet, entsprechend aber Bits die weiter entfernt sind eine geringere Wahrscheinlichkeit von Fehlern haben. Außerdem würden von diesen Annahmen abgeleitete Anforderungen an die einen Vorwärtsfehlerkorrektur (Forward Error Correction, FEC) Mechanismus unterschätzt werden: Um effektiv zu sein muss der FEC-Mechanismus typischerweise in der Lage sein mehrere Bitfehler in einem Paket zu erkennen oder zu korrigieren, was entsprechend dieser Berechnung überaus selten wäre:

$$\begin{aligned} \mathsf{P}(\textit{mehrere Bitfehler in Paket}) &= 1 - \Big(\mathsf{P}(\textit{kein Fehler}) \quad + \mathsf{P}(\textit{exakt ein Bitfehler})\Big) \\ &= 1 - \Big((1-p_b)^n \quad + \binom{n}{1} p_b (1-p_b)^{n-1}\Big) \end{aligned}$$

   Mit den Zahlen aus dem obigen Beispiel ergibt sich eine Wahrscheinlichkeit von $1.1 \cdot 10^{-8}$, während der typische Wert deutlich höher wäre. Die exakten Muster sind jedoch stark abhängig von der verwendeten Technologie.

[a] https://de.wikipedia.org/wiki/Geometrische_Verteilung

Übung 6 *Effizienz des Go-Back-N Protokolls*

1. Bestimmen sie die Effizienz des Go-Back-N-Mechanismus, wenn bei der Übertragung von Daten keine Fehler auftreten. Die Effizienz eines Protokolls ist das Verhältnis zwischen erreichtem Durchsatz (Achtung: Netto! Also Goodput[2], nicht Throughput) und dem theoretischen Maximum.
   - Die Größe des verwendeten Fensters ist w.

[2] https://de.wikipedia.org/wiki/Datendurchsatz

- Die Größe jedes Datenpakets ist n_d Bits
- die Größe der Acknowledgements ist n_a Bits.
- Bei jeder Datenübertragung entsteht eine Verzögerung τ.
- Die Datenrate des verwendeten Mediums ist R.

Fertigen sie eine Skizze an und geben sie die allgemeine Formel an.

2. Wie gross muss das Fenster mindestens sein, damit das Protokoll eine Effizienz von 1 erreicht?

Lösung

1. Ohne Skizze, dafür mit viel Text.

   Für die Berechnung der Effizienz reicht es, die benötigte Zeit einer Übertragung zu betrachten (sei `r` der erreichte Nettodurchsatz, `O` die Objektgröße und `t` Zeiten. Die Suffixe `t` und `p` stehen für theoretisch und praktisch):

   $E = \frac{B_p}{B_t} = \frac{Ot_p^{-1}}{Ot_t^{-1}} = \frac{t_t}{t_p}$

   Die theoretische minimale Zeit die benötigt wird um die Daten komplett zu übertragen ist $t_t = O/R + \tau$: Die Daten müssen komplett gesendet werden, sich durch das Medium ausbreiten und beim Empfänger ankommen.

   Die Zeit die bei Verwendung von Go-Back-N benötigt wird ist komplexer zu berechnen. Da es in diesem Szenario keine Fehler gibt werden die Daten in $n = \lceil \frac{O}{wn_d} \rceil$ Fenstern übertragen, wobei das letzte Fenster nicht notwendigerweise komplett gefüllt wird. Nachdem der Sender ein Fenster gesendet hat wartet er auf ein Acknowledgement. Entsprechend setzt die Gesamtzeit für die Übertragung wie folgt zusammen:

   $t_p = (n-1)(\frac{wn_d}{R} + t_{delay}) + (\frac{O \mod wn_d}{R}) + \tau = \frac{O}{R} + \tau + (n-1)t_{delay}$

   Die einzige Unbekannte hier ist t_{delay}, die Zeit die nach dem Senden auf ein Acknowledgement gewartet wird bevor weitere Daten übertragen werden können. Wir können dies bestimmen, in dem wir die Zeit berechnen die das Senden des ersten Pakets eines Fensters sowie dessen Acknowledgements benötigt:

   $t_{cont} = \frac{n_d}{R} + \tau + \frac{n_a}{R} + \tau = \frac{n_d+n_a}{R} + 2\tau$

   und die Übertragungsdauer der Daten des Fensters abziehen:

   $t_{delay} = t_{cont} - \frac{wn_d}{R} = \frac{n_d+n_a}{R} + 2\tau - \frac{wn_d}{R}$

   Durch Einsetzen erhalten wir:

   $t_p = \frac{O}{R} + \tau + (n-1)(\frac{n_d+n_a}{R} + 2\tau - \frac{wn_d}{R})$

2. Die obere Formel gilt nur, wenn der Sender tatsächlich warten muss, bevor mit dem Senden des zweiten Fensters begonnen werden kann. Dies ist

der Fall, wenn das Warten auf das Acknowledgement länger dauert, als das Senden der Daten des Fensters benötigt:

$t_{cont} = \frac{n_d+n_a}{R} + 2\tau - \frac{wn_d}{R} > \frac{wn_d}{R}$

Ist dies nicht der Fall, ist $t_{delay} = 0$, und die Formel für t_p vereinfacht sich:

$t_p = \frac{O}{R} + \tau = t_t \Rightarrow E = 1$

Übung 7 *TCP vs. UDP*

Diskutieren Sie die Vor- und Nachteile von UDP und TCP für die folgenden Anwendungsbeispiele. Welches Transportprotokoll ist jeweils sinnvoller?

1. Fernzugriff auf eine Datenbank
2. Videostreaming
3. Videotelefonie
4. Übertragung großer Mengen von Archivdaten
5. Multiplayer Gaming
6. Ein P2P-Overlay-Netzwerk

Lösung

Generell muss abgewägt werden, welche Anforderungen die Anwendungen haben. Die Verwendung von TCP hat Konsequenzen für den Netzwerkverkehr welche für die Anwendung angemessen sein können, aber auch ungewünscht sein können. UDP hingegen ist ein simples Protokoll, welches wenige Garantien zusätzlich zum zugrunde liegenden Netzwerkprotokoll macht, allerdings auch einen kleinen Overhead hat. Entsprechend lässt sich die Entscheidung zwischen TCP und UDP darauf reduzieren, ob man die Funktionen, welche TCP bereitstellt benötigt, oder nicht.
Diese Funktionen lassen sich größtenteils unter dem Kontext der Zuverlässigkeit sehen: Empfangsbestätigungen, Garantien über Nachrichtenreihenfolge, Fehlererkennung, sowie Fluß- und Überlastkontrolle stellen das Bereit, was Programmierer sich unter einem artigem Kanal vorstellen. Die Kosten dafür sind allerdings höhere Latenzen, insbesondere beim Verbindungsaufbau, eine klare 1:1 Kommunikation, sowie eine geringere Flexibilität. Diese Einschränkungen machen die Verwendung von TCP zu einer suboptimalen Wahl für **??**, da hier Flexibilität bzgl. der Kommunikationsmuster essentiell ist.
Diese Zuverlässigkeit macht TCP zu einem guten Kandidaten für **????**. In beiden Fällen ist wichtig, dass Gewissheit darüber herrscht, welche Daten angekommen sind und welche gegebenenfalls verloren gegangen sind. Darüber hinaus ist die

unveränderte Reihenfolge der Daten gewährleistet. Für die Übertragung großer Datenmengen könnte die fehlende Flexibilität je nach Anwendungsfall ein Nachteil sein, der aber im Allgemeinen den Nutzen nicht aufwiegt. In Extremfällen kann aber ein optimiertes Protokoll für den konkreten Anwendungsfall Sinn machen, trotz des hohen Entwicklungsaufwandes.
Für **????** ist charakteristisch, dass bei Datenverlust eine erneute Übertragung der Daten häufig nichtmehr praktikabel ist. Die Wiedergabe von Audio-/Videodaten lässt sich nur sehr eingeschränkt dehnen bis Wiederübertragungen empfangen werden. Da dies der zentrale Mechanismus von TCP ist um verlustbehaftete Kanäle zu kompensieren, ist dies entsprechend keine gute Wahl. Im Gegenteil: Der Versuch, Daten erneut zu versenden die bei Ankunft nicht mehr verwendet werden können verwendet Resourcen die dann an anderer Stelle fehlen können. Forwärtsfehlerkorrekturmechanismen können hier bessere Ergebnisse liefern.
In der Praxis wird für Anwendungen wie Netflix oder Zoom häufig dennoch TCP eingesetzt. Dies hat hauptsächlich zwei Gründe: Einserseits bietet TCP effektive Mechanismen zur Flußkontrolle welche erlauben die verfügbare Übertragungskapazität einzuschätzen. Darüber hinaus ist die Verwendung von TCP (und konkreter HTTP) häufig der einzig praktikable Mechanismus um sicherzustellen, dass Daten in den meisten Fällen nicht durch Firewalls abgefangen werden.
Ein Sonderfall ist **??**: Multiplayer Gaming. Die Echtzeitanforderungen sind hier häufig noch größer als bei Videotelefonie, gleichzeitig ist Zuverlässigkeit auch überaus wichtig. Anwendungen setzen daher häufig auf eine Kombination von UDP und TCP um den verschiedenen Anforderungen gerecht zu werden.

Übung 8 *3-Wege-Handshake*

Einem Chatbot[3] wurde folgende **Aufgabe** gestellt:

> **!** Erklären Sie den Prozess des TCP 3-Wege-Handshake und den Zweck jedes Schrittes. Beziehen Sie in Ihrer Erklärung die spezifischen Nachrichten ein, die zwischen dem Client und dem Server während des Handshakes ausgetauscht werden.

Die **Antwort** des Chatbots war die folgende:

[3] Diese Aufgabe wurde unter Zuhilfenahme von ChatGPT (`https://openai.com/blog/chatgpt/`) erstellt.

Der TCP-3-Wege-Handshake ist der Prozess, der verwendet wird, um eine zuverlässige, geordnete Verbindung zwischen zwei Geräten, typischerweise einem Client und einem Server, unter Verwendung des Transmission Control Protocol (TCP) herzustellen. Der Prozess besteht aus drei Schritten:

1. SYN (Synchronisieren): Der Client sendet eine Nachricht an den Server, in der das SYN-Flag auf 1 gesetzt ist, um anzuzeigen, dass er eine neue Verbindung aufbauen möchte. Diese Nachricht enthält auch die vom Client gewählte Anfangssequenznummer (ISN).
2. SYN-ACK (Synchronize-Acknowledge): Der Client empfängt die SYN-Nachricht und antwortet mit einer Nachricht, bei der sowohl das SYN- als auch das ACK-Flag auf 1 gesetzt sind. Diese Nachricht enthält die eigene ISN des Servers und bestätigt den Empfang der ISN des Clients.
3. ACK (Acknowledge): Der Server empfängt die SYN-ACK-Nachricht und sendet eine letzte Nachricht an den Client, in der das ACK-Flag auf 1 gesetzt ist und der Empfang der ISN des Clients bestätigt wird.

Der Zweck der SYN-Nachricht besteht darin, die Verbindung zu beenden und die Sequenznummern zwischen Client und Server zu synchronisieren. Die SYN-ACK-Nachricht dient dazu, den Empfang der SYN-Nachricht zu bestätigen und die eigene ISN des Clients zu übermitteln. Die abschließende ACK-Nachricht bestätigt, dass der Server die SYN-ACK-Nachricht erhalten hat und die Verbindung hergestellt ist.
Sobald der Drei-Wege-Handshake abgeschlossen ist, gilt die Verbindung als beendet, und es können keine Daten mehr zwischen Client und Server über die bestehende Verbindung ausgetauscht werden.

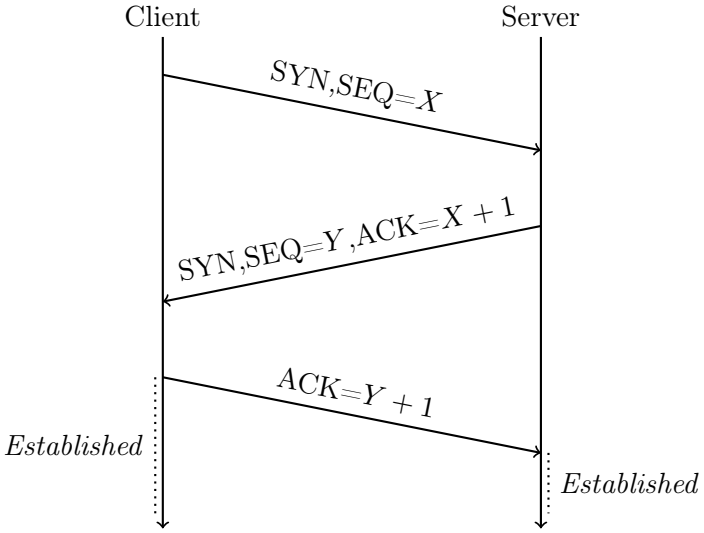
1. Leider haben sich einige Fehler in die Antwort des Chatbots eingeschlichen. Identifizieren Sie diese!
2. Visualisieren Sie den korrekten Ablauf des 3-Wege-Handshakes, inklusive der relevanten Flags die in den Nachrichten gesetzt sein müssen, sowie die verwendeten Sequenznummern/Acknowledgements.
3. In welchem Paket dürfen jeweils die ersten Nutzdaten in beiden Richtungen versandt werden?

Lösung

1. Die Antwort enthält subtile Fehler:
   - die SYN-ACK-Nachricht vom Server gesendet, nicht vom Client.
   - die endgültige ACK-Nachricht vom Server und nicht vom Client gesendet wird

- laut Beschreibung ist die Verbindung nach dem Handshake beendet. Tatsächlich ist das Gegenteil der Fall, die Verbindung wird danach zum Datenaustausch verwendet.

2. Das folgende Diagramm stellt den TCP-Handshake dar. Die Sequenznummern `X` und `Y` werden zufällig aus dem Sequenznummer-Raum (32 bit) ausgewählt.



3. Daten können versandt werden, sobald die Verbindung etabliert (*established*) ist. Sie können also auch im dritten und letzten Paket des Handshakes verwendet werden.

Übung 9 *TCP-Übertragung*

1. Berechnen Sie die Zeit zum Kopieren eines Objektes vom Server zum Client mit TCP. Nehmen Sie an, dass die TCP-Verbindung zwischen Server und Client gerade frisch aufgebaut wurde und das Objekt somit sofort übertragen werden kann. Nehmen Sie außerdem an, dass keine Paketverluste auftreten. Ignorieren Sie außerdem den Overhead von TCP-Headern sowie Protokollen tieferer Schichten.

   Gehen Sie von den folgenden Werten aus:

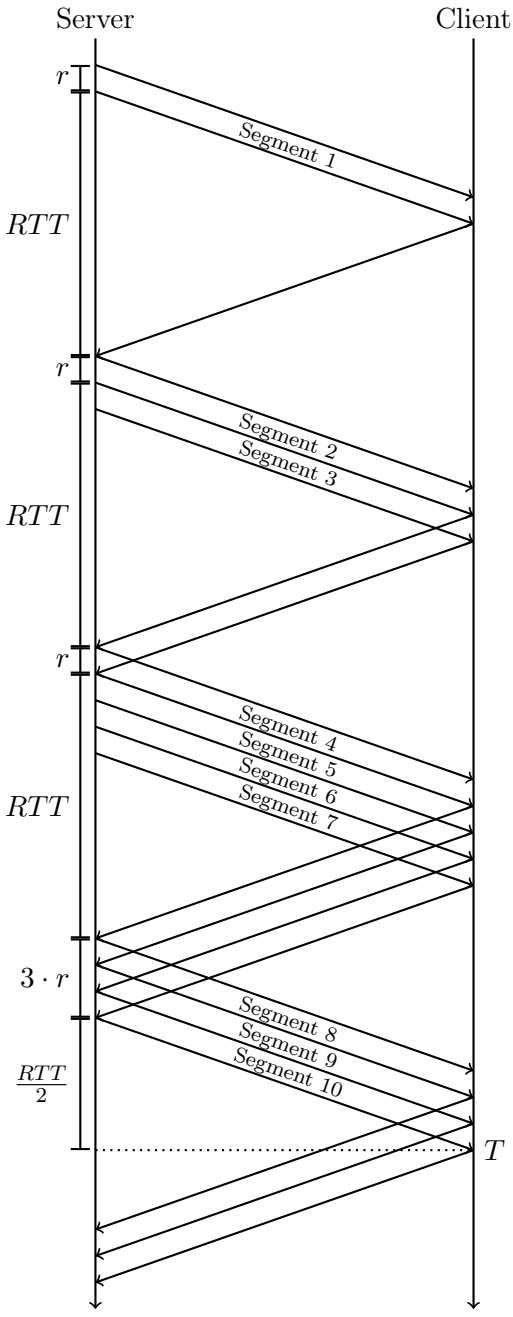
   - Objektgröße $O = 10\,\mathrm{kB}$
   - Segmentgröße $MSS = 1\,\mathrm{kB}$
   - Round-Trip-Time $RTT = 1\,\mathrm{s}$
   - Datenrate $R = 10\,\mathrm{kB\,s^{-1}}$

2. Diskutieren Sie, wie sich Änderungen der Vorgaben auf den Ablauf der Übertragung auswirken. Welchen Effekt hat eine Halbierung der Segmentgröße? Welchen Effekt hat eine Verdopplung der Objektgröße? Welchen Effekt hat eine Verdopplung der Datenrate?

3. Diskutieren Sie nun, wie sich der Ablauf der Übertragung verändert, wenn Paketverluste auftreten. Nehmen Sie beispielsweise an, dass jedes 2. Paket sicher verloren geht.

Lösung

1.
   - Anzahl Segmente $= \frac{O}{MSS} = 10$
   - Übertragungsverzögerung $= \frac{MSS}{R} = 0.1\,\mathrm{s}$


Server
Client
r
Segment 1
RTT
r
Segment 2
Segment 3
RTT
r
Segment 4
Segment 5
Segment 6
Segment 7
RTT
3 · r
Segment 8
Segment 9
Segment 10
RTT/2
T

$$
\begin{aligned}
T &= 3 \cdot (r + RTT) + 3 \cdot r + \frac{RTT}{2} \\
&= 3 \cdot (0.1\,\mathrm{s} + 1\,\mathrm{s}) + 3 \cdot 0.1\,\mathrm{s} + \frac{1\,\mathrm{s}}{2} \\
&= 4.1\,\mathrm{s}
\end{aligned}
$$

2. Eine Halbierung der Segmentgröße führt zu einer Halbierung der Übertragungsverzögerung je Segment bei Verdopplung der Anzahl an zu übertragenden Segmenten. Bei 20 Segmenten gibt es 5 Fenster, bis das Objekt übertragen wurde, wobei im letzten Fenster 5 Segmente übertragen werden.

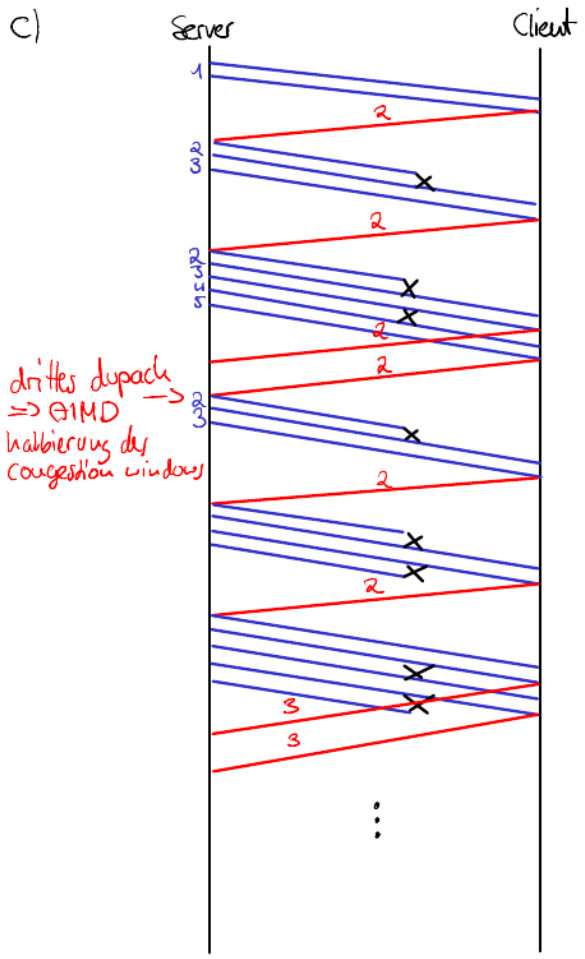
$$
\begin{aligned}
T &= 4 \cdot (r + RTT) + 5 \cdot r + \frac{RTT}{2} \\
&= 4 \cdot (0.05\,\mathrm{s} + 1\,\mathrm{s}) + 5 \cdot 0.05\,\mathrm{s} + \frac{1\,\mathrm{s}}{2} \\
&= 4.95\,\mathrm{s}
\end{aligned}
$$

   Entgegen die Intuition führt eine Verdopplung der Objektgröße nicht zu einer Verdopplung der Übertragungszeit. Zu Beginn der Übertragung ist die Zeit dominiert durch das Warten auf Acknowledgements (bei hinreichend großer RTT). Daher führt eine doppelte Objektgröße zu einer um ca. 30 % längeren Übertragung:

$$
T = 4 \cdot (r + RTT) + 5 \cdot r + \frac{RTT}{2} = 5.4\,\mathrm{s}
$$

   Dies unterstreicht den Nutzen von persistenten TCP Verbindungen, beispielsweise für HTTP, nicht nur um Verbindungsauf- und abbau zu sparen, sondern auch für die Anpassung an den Kanal.

3. Extremes Beispiel: Es geht immer genau jedes zweite Datenpaket verloren aber ACKs gehen nicht verloren.



Obwohl "nur"die Hälfte der Daten (d.h. jedes zweite Paket) verloren geht, kommt es zu extremen Verzögerungen bei der Übertragung. Bei häufigen Paketverlusten kommt TCP nie auf dem Slow-start hinaus. Für eine genaue Berechnung der Übertragungsdauer werden noch zusätzliche Informationen über die auftretenden Timeouts benötigt.

Übung 10 *TIME-WAIT*

RFC 1122[4] sieht vor, dass Sockets nach dem Schließen im `TIME-WAIT` Zustand verbleiben:

[4] `https://www.rfc-editor.org/rfc/rfc1122#page-88`

! When a connection is closed actively, it MUST linger in TIME-WAIT state for a time 2xMSL (Maximum Segment Lifetime). However, it MAY accept a new SYN from the remote TCP to reopen the connection directly from TIME-WAIT state, if it: [...]

Die MSL ist laut TCP 120 s, entsprechend verweilen Sockets für eine signifikante Zeit. Dies führt dazu, dass Anwendungen den Socket in diesem Zeitraum nicht verwenden können, da die Socket `Address already in use` ist.

Warum sieht TCP diesen Zustand vor? In welchem Fall ist das Verweilen des Sockets sinnvoll?

Lösung

Der `TIME-WAIT` Zustand beugt Problemen in zwei Fällen vor:

1. Ein verlorenes letztes Acknowledgement der TCP Verbindung: wird das letzte Paket des TCP-Teardowns nicht empfangen, verweilt die Gegenseite im `LAST_ACK` Zustand. Wäre der Socket nun schon geschlossen, wäre ein erneuter Verbindungsaufbau nicht möglich, da die Gegenseite kein `SYN` erwartet. Im `TIME-WAIT` Zustand kann der Socket auf ein wiederholtes `FIN` reagieren und das Acknowledgement erneut übertragen.
2. Die Verbindung kann beendet werden während noch Daten erwartet werden. Sind diese Daten nicht verloren, sondern nur verzögert, können diese eine erneute Verbindung des gleichen Sockets stören.

Übung 11 *Fluss- und Überlastkontrolle*

Beantworten Sie die folgenden Fragen im Kontext von Transportprotokollen:

1. Was ist der Unterschied zwischen Flusskontrolle und Überlastkontrolle?
2. Über welchen Mechanismus wird Flusskontrolle in TCP implementiert?
3. Über welchen Mechanismus wird Überlastkontrolle in TCP implementiert?
4. Inwiefern hängen die beiden Aspekte zusammen?

Lösung

1. Flusskontrolle bezeichnet den Versuch, den Empfänger nicht zu überlasten, während durch Überlastkontrolle versucht wird den Kanal nicht zu sehr auszulasten.
2. Der Empfänger kommuniziert in den Acknowledgements, wieviele Daten er maximal empfangen kann. Falls der Sender mehr Daten senden könnte, war-

tet er auf weitere Acknowledgements und ein gegebenenfalls größeres *advertised window*.

3. Der Sender erhöht kontinuierlich die Menge der Daten die zu einem bestimmten Zeitpunkt noch unbestätigt sein können. Erkennt der Sender durch ausbleibende Acknowledgements Datenverlust verkleinert er das Fenster, sodass sich dieses an der Kapazität des Kanals orientiert.

4. Beide Mechanismen regeln wieviele Daten der Sender maximal senden kann. Dabei wird die tatsächlich versandte Datenmenge durch das Minimum, also durch den größten Engpass bestimmt.