Moritz Hahn
Sarah Altenkrüger

1	2	3	Σ

Übungsblatt Nr. 7

(Abgabetermin 10.06.2024)

Aufgabe 7.1

1.

Die Übertragung von Daten direkt in IP-Paketen ist nicht ausreichend, da IP (Internet Protocol) selbst keine Mechanismen zur zuverlässigen Datenübertragung bietet. Hier sind einige Gründe dafür:

- **Fehlende Zuverlässigkeit**: IP-Pakete können verloren gehen, dupliziert oder in falscher Reihenfolge ankommen. IP garantiert nicht, dass Pakete erfolgreich zugestellt werden.
- **Keine Fehlerkorrektur**: IP enthält keine Mechanismen zur Fehlererkennung und -korrektur der übermittelten Daten.
- **Reihenfolge der Pakete**: IP garantiert nicht, dass Pakete in der Reihenfolge ankommen, in der sie gesendet wurden.
- **Keine Verbindungskontrolle**: IP ist ein verbindungsloses Protokoll, das keine dauerhafte Verbindung zwischen Sender und Empfänger aufrechterhält.

Aus diesen Gründen verwenden Anwendungen TCP (Transmission Control Protocol) oder UDP (User Datagram Protocol) als Transportprotokolle, um zusätzliche Funktionen wie Zuverlässigkeit, Fehlerkorrektur und Sequenzierung bereitzustellen.

2.

UDP (User Datagram Protocol) wird in bestimmten Anwendungsfällen gegenüber TCP bevorzugt, trotz seiner geringeren Zuverlässigkeit. Zwei Vorteile von UDP sind:

- **Geringere Latenz**: UDP ist schneller als TCP, da es keine Verbindung aufbaut und keine Zustellbestätigungen (ACKs) verwendet. Dies reduziert die Übertragungsverzögerung und ist besonders vorteilhaft für zeitkritische Anwendungen wie VoIP (Voice over IP), Online-Gaming und Echtzeit-Video-Streaming.
- **Einfachheit und geringerer Overhead**: UDP hat einen geringeren Protokolloverhead im Vergleich zu TCP. Es verwendet ein einfacheres Protokoll ohne komplexe Steuerungsmechanismen wie Verbindungsaufbau, Fehlerkorrektur und Flusskontrolle. Dies führt zu weniger Datenverkehr und weniger Ressourcenverbrauch.

3.

Ja, es ist möglich, dass eine Anwendung Daten zuverlässig austauscht, selbst wenn sie UDP verwendet. Allerdings muss die Zuverlässigkeit in diesem Fall von der Anwendung selbst implementiert werden. Hier sind einige Möglichkeiten, wie Anwendungen Zuverlässigkeit über UDP sicherstellen können:

- **Bestätigungsmechanismen (ACKs)**: Die Anwendung kann Empfangsbestätigungen für gesendete Pakete implementieren. Wenn der Sender keine Bestätigung erhält, sendet er das Paket erneut.
- **Sequenznummern**: Pakete können mit Sequenznummern versehen werden, sodass der Empfänger Pakete in der richtigen Reihenfolge zusammenfügen kann und doppelte Pakete erkennt.
- **Zeitüberschreitungen und Wiederholungen**: Wenn keine Bestätigung innerhalb einer bestimmten Zeitspanne empfangen wird, kann das Paket erneut gesendet werden.
- **Fehlerprüfung und -korrektur**: Mechanismen wie Prüfsummen oder Fehlerkorrekturcodes (z.B. Forward Error Correction) können verwendet werden, um sicherzustellen, dass empfangene Daten korrekt sind und gegebenenfalls korrigiert werden können.

Diese Techniken erfordern zusätzliche Implementierung und erhöhen den Komplexitätsgrad der Anwendung, bieten aber die Möglichkeit, die Flexibilität und Geschwindigkeit von UDP zu nutzen, während gleichzeitig ein gewisser Grad an Zuverlässigkeit erreicht wird.

4.

Ja, die UDP Segmente von Host A und Host B werden auf denselben Socket in Host C ge-demultiplext.

Dies liegt daran, dass UDP-Sockets in der Regel durch ein 2-Tupel bestehend aus der IP-Adresse und der Portnummer des Zielhosts identifiziert werden. In diesem Fall ist der Zielhost Host C mit der Portnummer 6832. Da beide Segmente an dieselbe IP-Adresse und dieselbe Portnummer gesendet werden, wird Host C diese Segmente auf denselben Socket weiterleiten. UDP bietet keine Mechanismen zur Unterscheidung der Segmente basierend auf den Absenderinformationen, sondern leitet sie einfach an den Zielport weiter.

5.

Ein UDP-Socket wird über ein 2-Tupel (IP-Adresse und Portnummer) identifiziert, während ein TCP-Socket über ein 4-Tupel (Quell-IP-Adresse, Quell-Portnummer, Ziel-IP-Adresse, Ziel-Portnummer) identifiziert wird. Dies liegt an den grundlegenden Unterschieden in der Funktionsweise und den Anforderungen der beiden Protokolle:

- **Verbindungsorientierung bei TCP**: TCP ist ein verbindungsorientiertes Protokoll, bei dem vor dem Datenaustausch eine Verbindung zwischen zwei Endpunkten hergestellt wird. Diese Verbindung wird durch die Kombination aus Quell- und Ziel-IP-Adresse sowie Quell- und Ziel-Portnummer eindeutig identifiziert. Das 4-Tupel ermöglicht es, mehrere parallele Verbindungen zwischen denselben Endpunkten zu unterscheiden.

- **Verbindungslosigkeit bei UDP**: UDP ist ein verbindungsloses Protokoll, das keine dauerhafte Verbindung zwischen Sender und Empfänger aufrechterhält. Da UDP keine Verbindung verwaltet und jedes Datagramm unabhängig behandelt wird, reicht es aus, das Datagramm durch die Ziel-IP-Adresse und die Ziel-Portnummer (2-Tupel) zu identifizieren. Dies ermöglicht eine einfache und schnelle Übermittlung der Daten ohne die Notwendigkeit einer komplexen Verbindungsverwaltung.
- **Effizienz und Einfachheit bei UDP**: Da UDP keine Verbindungen aufbaut oder verwaltet, ist es effizienter und einfacher zu implementieren. Die Identifikation über ein 2-Tupel reicht aus, um die Datagramme an den richtigen Zielsocket zu übermitteln, was dem Designprinzip von UDP als leichtgewichtiges und schnelles Protokoll entspricht.

Diese Unterschiede spiegeln die jeweiligen Einsatzgebiete und Designziele von TCP und UDP wider, wobei TCP eine zuverlässige und geordnete Datenübertragung für Anwendungen erfordert, die eine stabile Verbindung benötigen, während UDP für Anwendungen optimiert ist, die auf Geschwindigkeit und Effizienz angewiesen sind und eine gewisse Fehlertoleranz aufweisen.

7.2

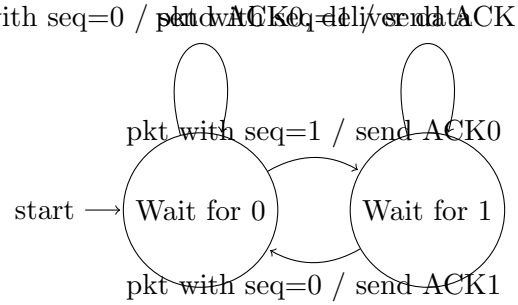
1.

Das RDT3.0 Protokoll (Stop-and-Wait) trifft folgende Annahmen über den zugrundeliegenden Übertragungskanal:

1. Der Kanal kann Pakete verlieren.
2. Der Kanal kann Pakete beschädigen.
3. Der Kanal kann Pakete verzögern.
4. Der Kanal liefert Pakete in der Reihenfolge, in der sie gesendet wurden.
5. Der Kanal kann ACKs (Bestätigungen) verlieren, beschädigen oder verzögern.

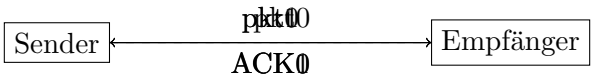
2.

Die finite state machine (FSM) der Empfängerseite für RDT3.0 sieht wie folgt aus:



3.

Nehmen wir an, dass der Übertragungskanal zusätzlich zu den in (1) genannten Annahmen die Paketreihenfolge ändern kann. Das Alternating-Bit Verfahren von RDT3.0 reicht für diesen Anwendungsfall nicht aus. Dies kann wie folgt gezeigt werden:



In diesem Beispiel: - Der Sender sendet $pkt0$ und der Empfänger bestätigt mit $ACK0$. - Der Sender sendet $pkt1$ und der Empfänger bestätigt mit $ACK1$. - Angenommen, $pkt0$ wird erneut gesendet aufgrund einer Paketverlustsituation oder Wiederholungsanforderung und kommt nach $pkt1$ an.

Der Empfänger sieht $pkt0$ erneut, sendet jedoch $ACK0$. Der Sender könnte dies als eine Anforderung zur Wiederholung des letzten gesendeten Pakets missverstehen und $pkt1$ erneut senden, was zu doppelten Paketen beim Empfänger führt und die Reihenfolge der Daten durcheinander bringt. Dies zeigt, dass das Alternating-Bit Verfahren bei Paketreihenfolgeänderungen nicht zuverlässig ist.

7.3

1.

Die Zeit d_{SAW} für die Übertragung eines Pakets umfasst die Zeit, die benötigt wird, um das Paket zu senden, sowie die Round-Trip-Time (RTT):

- **Paketgröße** L: $4000\,\text{B} = 4000 \times 8 = 32000$ Bits
- **Übertragungsgeschwindigkeit** C: $10\,\text{Gbit}\,\text{s}^{-1} = 10 \times 10^9$ Bits pro Sekunde
- **Round-Trip-Time (RTT)**: $20\,\text{ms} = 20 \times 10^{-3}$ Sekunden

Die Übertragungszeit t_{send} für ein Paket ist:

$$t_{\text{send}} = \frac{L}{C} = \frac{32000}{10 \times 10^9} = 3.2 \times 10^{-6} \text{ Sekunden} = 3.2 \text{ Mikrosekunden}$$

Da die RTT die Zeit ist, die vergeht, bis ein ACK empfangen wird, beträgt die gesamte Zeit d_{SAW} für ein Paket:

$$d_{\text{SAW}} = t_{\text{send}} + \text{RTT} = 3.2 \text{ Mikrosekunden} + 20 \text{ Millisekunden} = 20.0032 \text{ Millisekunden}$$

2.

Die Auslastung des Kanals U_{SAW} gibt an, wie viel Prozent der Zeit der Kanal mit der Datenübertragung beschäftigt ist. Dies ist das Verhältnis der Sendezeit zur Gesamtzeit:

$$U_{\text{SAW}} = \frac{t_{\text{send}}}{d_{\text{SAW}}} = \frac{3.2 \times 10^{-6}}{20.0032 \times 10^{-3}} \approx 0.00016$$

In Prozent ausgedrückt:

$$U_{\text{SAW}} \approx 0.016\%$$

3.

Der effektive Durchsatz R_{SAW} ist die Anzahl der übertragenen Bits pro Sekunde, die tatsächlich erfolgreich empfangen werden:

$$R_{\text{SAW}} = \frac{L}{d_{\text{SAW}}} = \frac{32000 \text{ Bits}}{20.0032 \times 10^{-3} \text{ Sekunden}} \approx 1.599 \times 10^6 \text{ Bits pro Sekunde} = 1.599 \text{ Mbps}$$

Zusammengefasst:

- Die Zeit d_{SAW} für die Übertragung eines Pakets über den $10\,\text{Gbit}\,\text{s}^{-1}$-Link beträgt $20.0032\,\text{ms}$.
- Die Auslastung U_{SAW} des Kanals beträgt $0.016\,\%$.
- Der effektive Durchsatz R_{SAW} der Verbindung beträgt $1.599\,\text{Mbps}$.

4.

Um die Auslastung U_{GBN} von mehr als 90% zu erreichen, muss die Fenstergröße N so gewählt werden, dass der Sender immer Pakete im Pipelining-Modus sendet. Die Auslastung U_{GBN} kann berechnet werden mit:

$$U_{\text{GBN}} = \frac{N \cdot t_{\text{send}}}{d_{\text{GBN}}}$$

Hier ist d_{GBN} die Zeit, die für eine vollständige RTT benötigt wird, also:

$$d_{\text{GBN}} = \text{RTT}$$

Da die Paketübertragungszeit t_{send} gleich 3.2 Mikrosekunden beträgt, und wir eine Auslastung von mehr als 90% ($U_{\text{GBN}} > 0.90$) erreichen wollen, ergibt sich:

$$0.90 < \frac{N \cdot 3.2 \times 10^{-6} \text{ Sekunden}}{20 \times 10^{-3} \text{ Sekunden}}$$

Daraus folgt:

$$N > \frac{0.90 \times 20 \times 10^{-3}}{3.2 \times 10^{-6}}$$

$$N > \frac{0.018}{3.2 \times 10^{-6}}$$

$$N > 5625$$

Die minimale Fenstergröße N, um eine Auslastung von mehr als 90% zu erreichen, beträgt also:

$$N \geq 5626$$

5.

Das Go-Back-N (GBN) Protokoll ist ineffizient, wenn Paketverluste auftreten, da es bei Verlust eines einzelnen Pakets alle nachfolgenden Pakete erneut senden muss, selbst wenn diese korrekt empfangen wurden. Dies führt zu einer hohen Anzahl redundanter Übertragungen und verringert somit den effektiven Durchsatz, insbesondere bei schlechten Netzwerkbedingungen oder hoher Paketverlustquote.

6. Verbesserung des GBN-Protokolls: Selective Repeat (SR)

Das GBN-Protokoll kann durch das Selective Repeat (SR) Protokoll verbessert werden, um die Effizienz bei häufigen Paketverlusten zu erhöhen. Bei Selective Repeat werden nur die tatsächlich verlorenen oder beschädigten Pakete erneut gesendet, anstatt alle Pakete nach dem verlorenen Paket. Dies wird erreicht durch:

- **Individuelle Bestätigungen (ACKs)**: Der Empfänger sendet ACKs für jedes korrekt empfangene Paket, und der Sender speichert eine Kopie jedes gesendeten Pakets im Sende-Puffer, bis eine Bestätigung empfangen wird.
- **Paketpufferung beim Empfänger**: Der Empfänger speichert Pakete, die korrekt empfangen werden, auch wenn sie außerhalb der Reihenfolge ankommen, und fügt sie in der richtigen Reihenfolge zusammen, sobald alle fehlenden Pakete empfangen wurden.
- **Wiederholtes Senden spezifischer Pakete**: Der Sender wiederholt nur die Übertragung der Pakete, für die innerhalb einer bestimmten Zeitspanne keine ACKs empfangen wurden.

Diese Mechanismen minimieren die Anzahl der redundanten Übertragungen und erhöhen den effektiven Durchsatz, insbesondere in Netzwerken mit hoher Fehlerrate, indem sie die Übertragungskapazität effizienter nutzen.