TCP/IP UND UDP

Larcher, Alexander

Inhalt

Transportschicht von TCP/IP

- Bisher: wie gelangen Pakete von einem Rechner zu einem anderen Rechner (= Host)
- Eigentliche Kommunikation: erfolgt zwischen *Anwendungen* (Schicht 5-6-7 ISO/OSI)
    - heutzutage hauptsächlich HTTP(s) über Browser, aber auch E-Mail, SSH/Telnet etc.
    - sind Prozesse auf den jeweiligen Betriebssystemen
- Voraussetzung: *End-zu-End-Protokolle*
    - Mittler zwischen der Vermittlungsschicht (Schicht 3 ISO/OSI) und den Anwendungen

1. Aufgaben der Transportschicht

- Prozess A auf Rechner X erwartet *transparente Kommunikation* mit Prozess B auf Rechner Y
- *Segmentierung* von Datenströmen unterschiedlicher Anwendungen bzw. Anwendungsinstanzen
    - = Logistik im Straßen- oder Postverkehr: 1.) Daten werden beim Sender zerlegt, 2.) unter Beachtung aller Prozesse möglichst fair verschickt und 3.) beim Empfänger wieder vereinigt
- Bereitstellung *verbindungsloser* und *verbindungsorientierter* Transportmechanismen
- Mechanismen zur *Stau- und Flusskontrolle*

1.1. Segmentierung

- Segmentierung (= *Multiplexing*) von Datenströmen unterschiedlicher Anwendungen (Browser, Chat, Email, ...)
- Segmente werden in unabhängigen IP-Paketen zum Empfänger geroutet
- Empfänger muss die Segmente den einzelnen Datenströmen zuordnen und an die jeweilige Anwendung weiterreichen

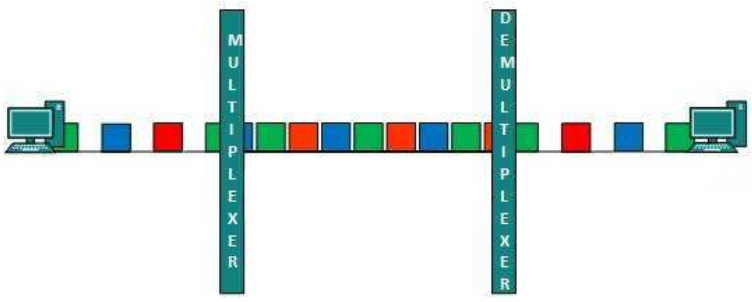


Abbildung 1: Segmentierung

1.2. Transportdienste

- *Verbindungslos* (Best Effort):
    - Segmente aus Sicht der Transportschicht voneinander unabhängig
    - Keine Sequenznummer, keine Übertragungswiederholung, keine Garantie der richtigen Reihenfolge
- *Verbindungsorientiert*:
    - Übertragungswiederholung bei Fehlern
    - Garantie der richtigen Reihenfolge einzelner Segmente

1.3. Stau- und Flusskontrolle

- Flusskontrolle: Schutz des Empfängers vor zu vielen Daten

- Überlastkontrolle: Schutz des Netzes vor zu vielen Daten

2. Adressen der Transportschicht

- Jedes Segment bekommt einen Header
  - Header enthält u.a. den Quellport und den Zielport
- Durch die Kombination von Header, IP-Adressen und verwendetem Transportprotokoll kann eine Anwendung eindeutig identifiziert werden
  - 5 Tupel (Transportprotokoll UDP/TCP, Quellport, Zielport, Quell-IP, Ziel-IP)

3. Verbindungsloses Protokoll - UDP (User Datagram Protocol)

3.1. Prinzip

- Verpacken, Verschicken, Vergessen
- Multiplexer zwischen verschiedenen Anwendungen und dem IP-Protokoll

3.2. Eigenschaften

- *Verbindungslos*, d.h. keine Verzögerung durch Verbindungsaufbau oder -abbau wie bei TCP
- *Kaum Overhead*, daher gut geeignet für Echtzeitanwendungen (Streaming, VoIP, Online Spiele)
- *Keine Garantie auf Zustellung*, *Vertauschung der Reihenfolge* aber auch *Verdoppelungen* möglich
- *Keine Stau- und Flusskontrolle* auf Transportebene
- *Eingeschränkte End-zu-End-Kontrolle*
  - Prüfsumme über Daten, bei Fehler wird das Datagramm verworfen
  - Datenlänge laut IP-Standard bis 64K, Fragmentierung allerdings problematisch (schon ein einziges fehlendes bzw. fehlerhaftes Fragment bewirkt Verwerfen aller Daten)
  - *Gängige Lösung*: Kleine Datenmengen bis 1,4 KByte[1] finden in nur einem PPP/ETH-Frame (Layer 2) Platz, daher größere Gewähr einer korrekten Zustellung
- Auch *Multicast* + *Broadcast* (Versand an mehrere bzw. alle anderen Teilnehmer)
- Im Loopback-Netz (127.0.0.0/8) sind diese Einschränkungen hinfällig

3.3. Der UDP Header

Abbildung 2: UDP Header

- Datenteil kann auch leer sein (Signalisierung)
- Prüfsumme kann optional nicht gesetzt werden (Wert 0, kommt aber sehr selten vor)
- Pakete, bei denen die Prüfsumme nicht stimmt, werden vom Empfänger automatisch verworfen
- Maximale Paketlänge 64K (16 Bit)

[1] ETH/PPP MTU=1500 – 20 Byte IP-Header – 8 Byte UDP-Header

4. UDP-Lite (RFC 3828)

- Wenn bei UDP auch nur 1 Bit fehlerhaft ist wird das ganze Datagramm verworfen
- Applikationen wie **VoIP** oder **Video/Audio-Streaming** können allerdings auch mit teilweise richtigen Daten etwas anfangen.
- Daher:
    - Ersetzung Header-Feld **Länge** durch **Checksum Coverage Length**.
        - Darin wird angegeben, ob die Checksum **nur** über den **Header** oder über das **ganze Datagram** berechnet wird (=UDP).
        - Paket-Länge geht nicht verloren, da sie aus der IP-Paket-Länge – 8 Bytes UDP-Header rekonstruiert werden kann.
    - Zusätzliche Möglichkeit: **Deaktivierung FCS/CRC** im **Data Link Layer**
        - Leider nicht allgemein unterstützt, da Verletzung der Standards[2]
        - Netzlayer IP-Checksumme bezieht sich nur auf den IP-Header, daher ok
- Hat sich aber nicht wirklich durchgesetzt:
    - Im **Linux-Kernel ab 2.6.20** und **FreeBSD 10.1-RELEASE** fix implementiert
    - Auf Windows nur über externe **WULL: A WINDOWS UDP-LITE LIBRARY**

5. Verbindungsorientiertes Protokoll - TCP (Transmission Control Protocol)

- Viele Anwendungen benötigen eine *zuverlässige Verbindung*
- TCP realisiert diese mit einem *verbindungsorientierten* Byte-Strom
- Inzwischen über QUIC neuer Ansatz eines verschlüsseltens TCPs auf UDP-Basis: https://de.wikipedia.org/wiki/QUIC

5.1. Prinzip

- Verbindungsaufbau, Datenübertragung, Verbindungsabbau
- Multiplexer zwischen verschiedenen Anwendungen und dem IP-Protokoll (analog zu UDP)

5.2. Eigenschaften

- Verbindungorientiert
- Zuverlässige Übertragung
- Garantie für Reihenfolge
- End-zu-Endkontrolle
- Zweiweg-Kommunikation (Vollduplex-Byte-Strom)
- Flusskontrolle
- Überlastkontrolle

5.3. Fehlende Eigenschaften

- Kein Broadcast/Multicast
- Keine Echtzeit

5.4. Segmentierung

- Aus Sicht von Anwendung: TCP stellt Byte-Strom dar
    - Einzelne Bytes können gelesen oder geschrieben werden
- Übertragung: Pakete werden von IP übertragen

[2] https://stackoverflow.com/questions/22101650/how-can-i-receive-the-wrong-ethernet-frames-and-disable-the-crc-fcs-calcul

- Wenig Sinn jedes Byte in eigenem IP-Paket zu versenden
- Schlechtes Verhältnis zwischen Nutzdaten und Header

- Lösung: Byte-Strom umsetzen in eine Folge von Segmenten

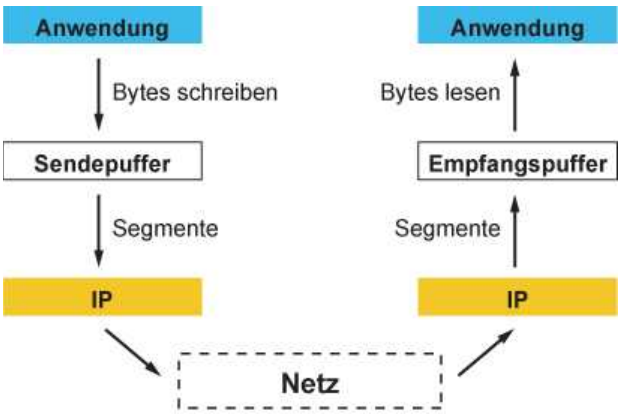


Abbildung 3: Segmentierung

- Große Segmente:
  - gutes Verhältnis von Nutzdaten zu Header-Daten
  - lange Wartezeiten
  - Größe ist beschränkt
  - Segmente bis zu einer Größe von 1,4 KByte finden perfekt in IP-Paketen und PPP/ETH-Frames Platz, verhindern Zersplitterung[3]
- Kriterien für das Versenden des nächsten Segments:
  - Maximale Segmentgröße erreicht
  - Ablauf des Timers für Aufsammeln
  - Push-Operation seitens der Anwendung

5.5. Kommunikation zwischen Prozessen

- Ein Prozess muss folgende Informationen an TCP weitergeben:
  - **Source Address:** Network + Host + Port des Senders
  - **Destination Address:** Network + Host + Port des Empfängers
  - **Next Packet Sequence Number:** Nummer des nächsten Pakets
  - **Current Buffer Size:** Buffergröße des Senders
  - **Next Write Position:** Stelle im Buffer, wo als nächstes Daten geschrieben werden
  - **Next Read Position:** Stelle im Buffer, wo als nächstes Daten gelesen werden müssen, um das nächste Segment zusammenzustellen
  - **Timeout/Flag:** Gibt die Zeit an, nach der nicht zugestellte Daten erneut gesendet werden müssen (un-acknowledged). Flag dient zum Synchronisieren von Prozess und TCP (z.B. Semaphoren) für z.B. Statusmeldungen.

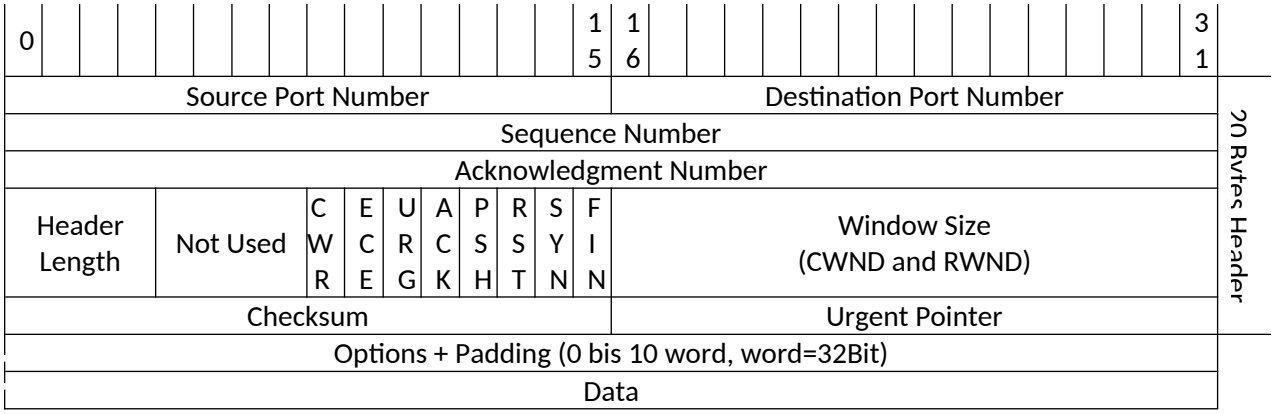
5.6. Bufferung:

- **Anwendungen** geben **Daten** in den **Puffer** von TCP. TCP kann sie direkt senden oder warten, bis genügend Daten zusammenkommen.
- Es bestehen folgende **zwei Ausnahmen**:

[3] https://de.wikipedia.org/wiki/Transmission_Control_Protocol#TCP-/IP-Segment-Gr%C3%B6%C3%9Fe

- Die Anwendung **deaktiviert die Pufferung**: Flag **PUSH** wird auf 1 gesetzt und die Daten umgehend verschickt.
- Die Anwendung **kennzeichnet dringende** (wichtige) **Daten**: Flag **URG** auf 1, wobei ein Verweis auf das Ende der umgehend verschickten Daten gesetzt wird (*Urgent Pointer*, siehe unten).

5.7. Der TCP Header



- **Source Port Number (16 Bit):** Port-Nummer des Senders
- **Destination Port Number (16 Bit):** Port-Nummer des Empfängers
- **Sequence Number (32 Bit):** Ansteigende Sequenznummer
- **Acknowledgment Number (32 Bit):** Wird nur verwendet, wenn ACK=1. Dann bestätigt es den korrekten Erhalt eines Teils der Daten, indem die nächste Sequenznummer, die angefordert wird, angegeben wird.
- **Header Length (4 Bit):** Gibt die Länge des Headers in 32Bit-Words an. Von 5 – 15 words (20 – 60 Bytes) an.
- **Not Used (4 Bit):** auf 0 gesetzt
- **Flags (8 Bit):** für die Kontrolle des Protokolls (z.B. Synchronisation)
  - **CWR: Congestion Window Reduced**, auf 1 gesetzt, wird der Sender die Geschwindigkeit verringern, wenn er ein Paket mit ECE=1 erhält
  - **ECE: ECN-Echo**, auf 1 gesetzt, zeigt es an, dass der Host **Explicit Congestion Notification** während des 3-way Handshakes unterstützt
  - **URG:** dringende Daten sind enthalten, **Urgent Pointer** muss gelesen werden
  - **ACK:** auf 1 gesetzt, ist ein Paket angekommen. Acknowledgment Number muss gelesen werden
  - **PSH: push,** auf 1 gesetzt, bedeutet es, dass die eingehenden Daten nicht gepuffert werden sollen, sondern gleich an den Application-Layer gegeben werden sollen
  - **RST: reset,** auf 1 gesetzt, zeigt es eine fehlerhafte Verbindung an. Wird in Zusammenhang mit ACK zum Beenden von Verbindungen verwendet.
  - **SYN: synchronize sequence number**, auf 1 gesetzt, möchte der Sender eine Verbindung aufbauen. Bei Sequence number wird eine initial Seq Num ISN eingetragen. Der Empfänger antwortet mit SYN-ACK, SYN=1
  - **FIN: final,** auf 1 gesetzt, hat der Sender keine Daten mehr zum Senden und möchte die Verbindung abbrechen. Der Empfänger antwortet mit FIN-ACK. Empfänger hat Kanal noch geöffnet (Problem: Halboffene Verbindungen)
- **Window Size (16 Bit):** Empfänger gibt damit an, wie viele Daten er empfangen kann.
- **Checksum (16 Bit):** gleiche Berechnung wie bei UDP, allerdings verpflichtend.

- **Urgent Pointer (16 Bit):** Wenn URG=1, wird darin der Offset angegeben, der zur Sequence Number addiert wird, um zum letzten dringenden Byte im Segment zu gelangen[4].
- **Options + Padding (0 – 10 word):** Wichtigste Option ist die Angabe der minimalen Segmentgröße, die empfangen werden kann. **536**([1]) Bytes für Daten und 20 Bytes für Header ist default, falls nichts angegeben. 556 Bytes müssen also akzeptiert werden. ([1]) 576 Bytes: die kleinste MTU bei IP die nicht fragmentiert werden muss (Waggon bei Zuganalogie) 576 – 20 -20 = **536** Bytes (TCP-Header, IP-Header)
- **Data:** Zu sendende oder zu empfangende Daten
- **Maximale Segmentgröße:** 1500 – 20 – 20 = **1460 Bytes** (ETH/PPP MTU, IP-Header, TCP-Header) (ganzer Zug bei Zuganalogie)

5.8. Verbindungsablauf TCP

- Verbindungsphasen:
  - Verbindungsaufbau (3 Way Handshake)
  - Datenübertragung
  - Verbindungsabbau (Teardown)

5.9. Verbindungsaufbau

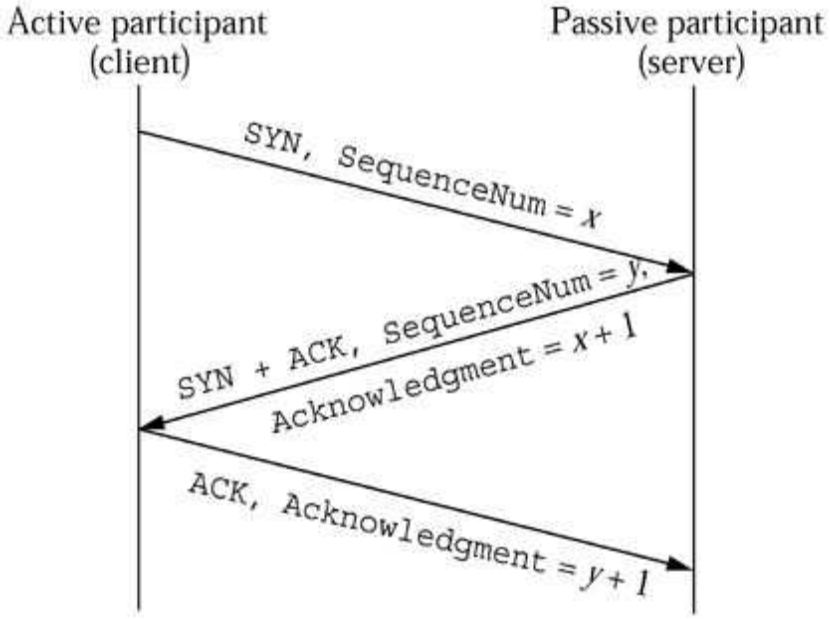


Abbildung 4: Three Way Handshake

1. Der Client schickt ein TCP-Segment mit dem Flag SYN=1 und einer zufällig gewählten Sequenznummer x.
2. Der Server bestätigt den Eingang durch ein TCP-Segment mit den Flags SYN=1 und ACK=1. Gleichzeitig inkrementiert er die Sequenznummer um 1 (x+1) und setzt seinerseits eine zufällige Sequenznummer y.
3. Der Client erhält die Bestätigung und schickt daraufhin ebenfalls eine Bestätigung mit ACK=1 und y+1 als „Acknowledge Number“.

- Gibt es auf Host 2 keinen Prozess der auf den angefragten Port hört, antwortet der Server im Schritt 2 mit der Flag RST=1. Die Verbindung wird direkt unterbrochen.

[4] Leider implementationsabhängig (Windows, GNU/Linux, BSD) unterschiedlich interpretiert

5.10. Verbindungsabbau

- Eine Verbindung wird in 3 Schritten abgebrochen und muss wieder von beiden Seiten bestätigt werden.
    1. Host 1 schickt FIN.F = 1 und SEQ.N = X
    2. Host 2 schickt ACK.F = 1 und ACK.N = X+1 und zusätzlich ACK.F = 1, ACK.N = X+1, FIN.F = 1 und SEQ.N = Y
    3. Host 1 schickt ACK.F = 1 und ACK.N = Y+1
- Ein Problem das Auftreten kann, ist eine halboffene Verbindung, wenn ein Host abbricht, aber der andere die Nachricht noch nicht erhalten hat. Nach einem Timeout kann dann jedoch ein Segment mit RST = 1 gesendet werden. Damit wird die Verbindung unmittelbar unterbrochen.

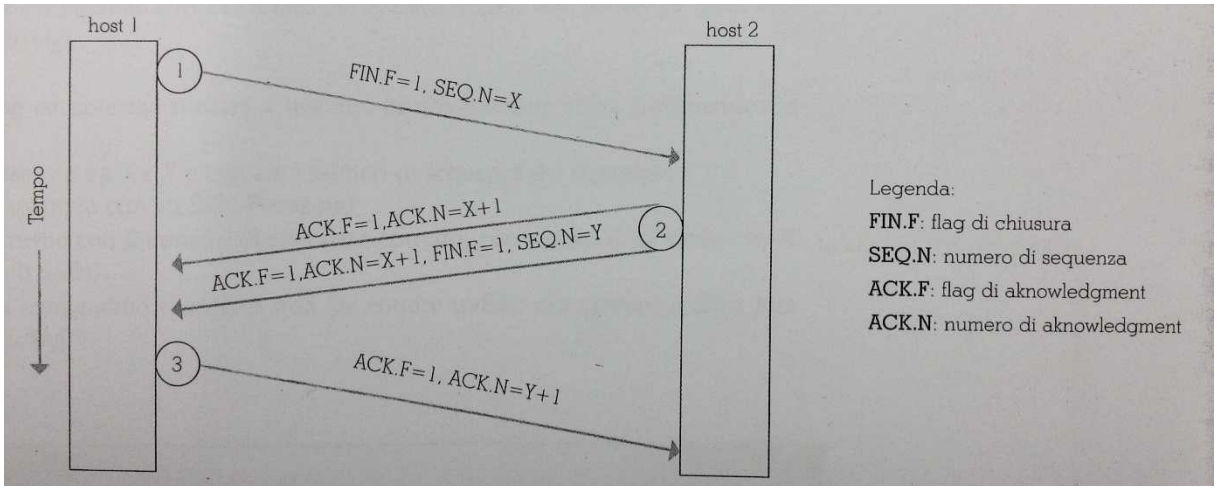


Abbildung 5: Verbindungsabbau

5.11. Stop and Wait Verfahren

- Sender wartet nach der Übertragung eines Segments auf eine Bestätigung, bevor er das nächste Segment überträgt.
- Kommt innerhalb der Wartezeit keine Bestätigung (Ablauf des *Retransmission-Timers*), überträgt der Sender das Segment erneut.
- **Allgemein**: Wenn Verbindung keine Datenübertragung mehr zulässt, kommt *Connection-Timer* zur Anwendung. Eine der beiden Gegenstellen sendet ein Paket mit RST=1
- **Contra:**
- Sehr ineffizient und viel Bandbreite bleibt ungenutzt (Netzwerk ist schlecht ausgelastet)
- Situation bei einer Verbindung im Internet noch verschärft wegen großer Antwortzeiten

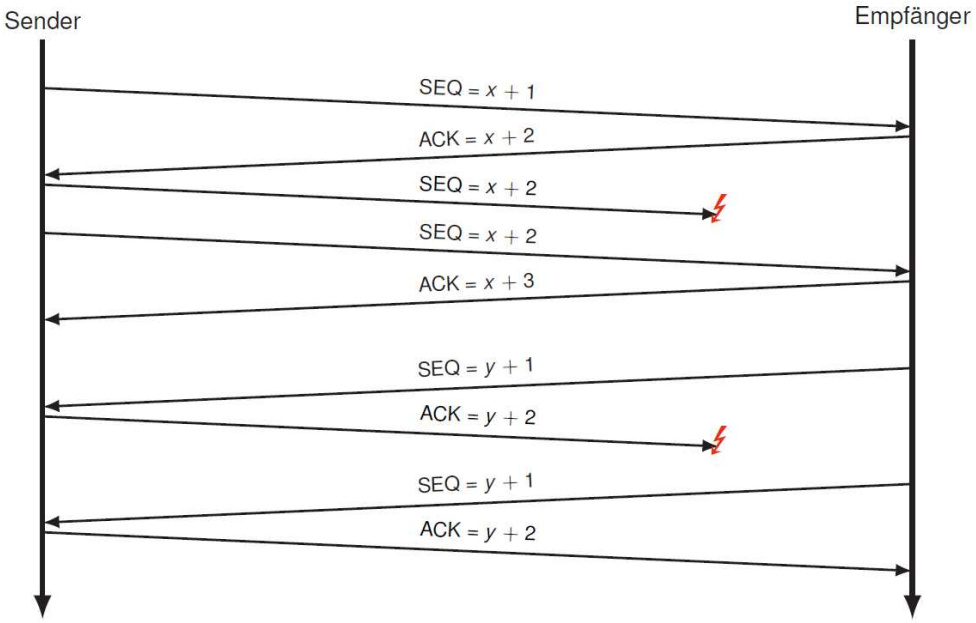


Abbildung 6: Stop and Wait Verfahren

5.12. Sliding-Window Verfahren

- Idee: Teile dem Sender mit, wie viele Segmente nacheinander übertragen werden dürfen ohne auf eine Bestätigung warten zu müssen

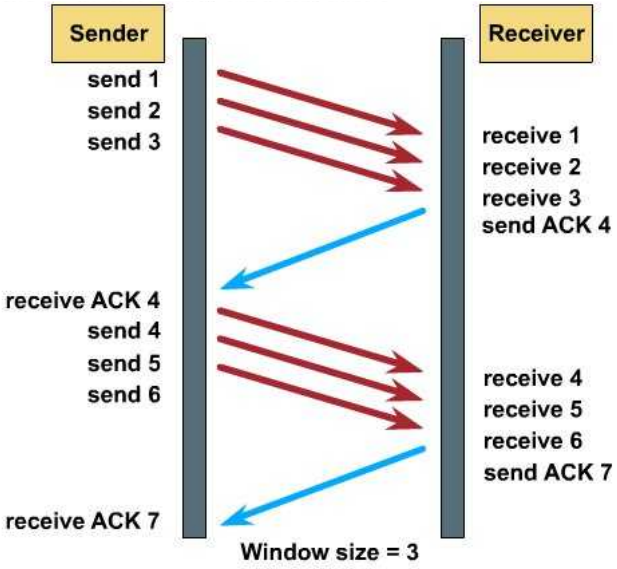


Abbildung 7: Sliding Window

- Die Fenstergröße kann und wird immer weiter vergrößert sofern alles gut läuft.
- Dabei läuft man aber auf zwei Gefahren zu:
  - Der Empfänger kann die Menge der Nachrichten nicht mehr aufnehmen
  - Das Netzwerk kann die Menge der Nachrichten nicht mehr transportieren
- Die Lösungen für diese Gefahren werden in den nächsten zwei Punkten abgehandelt. Für beide Lösungen spielt das Headerfeld „Window Size“ eine sehr wichtige Rolle.

5.13. TCP Flusskontrolle (Flow Control) - Überlastungskontrolle für den Empfänger

- Gefahr 1:
  - Empfängeranwendung kann die ankommenden Daten nicht schnell genug lesen
  - Empfangspuffer füllt sich bis kein Platz mehr ist
  - Eingehende Daten können nicht mehr bestätigt werden und alles muss nochmal gesendet werden
  - Viel unnötiger Netzverkehr entsteht
- Lösung:
  - Empfänger teilt dem Sender mit wie viele Bytes noch aufgenommen werden können
  - Dies teilt er im Feld Window Size mit.
    - Man spricht hier vom „Receiver Window“ (RWND) oder „Advertised Window“
  - Damit Sliding Window funktioniert, muss folgendes gewährleistet sein:
    - In jedem ACK des Empfängers gibt dieser seine Advertised Window Size an. Dies ist die Anzahl der Bytes die er in diesem Moment in der Lage ist zu empfangen
    - Der Sender darf nicht mehr Bytes übertragen als im letzten ACK angegeben
    - Der Empfänger muss alle Bytes annehmen können, die er im letzten ACK angegeben hat.
  - Die Größe des Sliding Windows entspricht üblicherweise dem freien Platz im Puffer. Ist der Puffer voll wird ACK mit Window Size = 0 gesendet.
  - Auf Layer 4 gibt es kein NACK wie auf Layer 2, aber es gibt ein Timeout. Nach Ablauf des Timeouts wird der Sender versuchen das Segment erneut zu senden. Der Sender muss also wissen welche Segmente übertragen aber noch nicht bestätigt wurden.
  - Dieser Mechanismus wird „Go back N“ genannt.

5.14. TCP Staukontrolle (Congestion Control) – Überlastungskontrolle für das Netzwerk

- Gefahr 2:
  - Durch Netzüberlastung können Segmente verloren gehen.
  - TCP reagiert darauf mit erhöhter Aktivität indem es die Segmente erneut sendet
  - Damit verschlimmert sich eine Überlast im Netz noch weiter, und es kann zu einem Kollaps kommen
- Lösung:
  - Schätzung der Netzbelastung durch den Sender
  - Der Sender steuert auf der Basis der beobachteten Paketübertragungen und Paketverluste seine Übertragungsgeschwindigkeit
    - Der Sender lotet Grenzen des Netzes aus in dem er die Window Size möglichst weit erhöht. In diesem Fall spricht man nun vom Congestion Window
    - Stößt er auf ein Limit (weil das Netzwerk oder der Empfänger die Daten nicht mehr übertragen/annehmen können) nimmt er die Rate zurück und nähert sich dem Optimum wieder von unten an.

- Das Congestion Window wird maximal so groß wie das Receiver Window des Empfängers. Aber es kann auch durchwegs kleiner bleiben weil das Netzwerk die Übertragung nicht mehr unterstützt.
- Um das Netz nicht zu überlasten ist es wichtig, dass nach Überschreitung des Limits schnell wieder auf einen sicheren Wert zurückgegangen wird.
- Umgekehrt kann das Herantasten an das Optimum langsam erfolgen.

- Die Priorität liegt auf dem Schutz des Netzes vor Überlast.

5.14.1. *Arten der Staukontrolle*

1. Slow Start/Congestion Avoidance (Van-Jacobsen-Algorithmus):

- Viele Variationen: TCP Tahoe, TCP Vegas, TCP Reno, TCP New Reno…
- Exponentielle Steigerung der Congestion Window Size bis zum ersten Time Out
  - 1,2,4,8,16,32,64…
- Beim ersten Time Out wird der Threshold (ssthresh) für die nächste Übertragung berechnet und wieder mit einer Congestion Window Size (cwnd) von 1 begonnen
  - ssthresh = cwnd/2 ssthresh = 64/2 = 32
  - cwnd = 1
- Nun wird das Congestion Window wider bis zum Erreichen des Threshold exponentiell vergrößert.
  - 1,2,4,8,16,32,33,34,35,36…
- Ab Erreichen des Threshold wird die cwnd nur mehr linear erhöht. Diese lineare Steigerung nennt man Congestion Avoidance.
- Werden sogenannte DupAcks (Duplicate Acks) empfangen kann man bereits erahnen, dass das Netz langsam an seine Grenzen kommt.
- **Ursprünglich** (heute überholt): Kommt es zum Timeout wird der neue Threshold berechnet (cwnd/2) und das cwnd wieder auf 1 zurückgesetzt (= ineffizient).
- **Fast-Retransmit/Fast-Recovery**: Beim dritten DupAck werden fehlende Pakete sofort nachgeschickt, es wird auf keinen Timer gewartet (*fast retransmit*). Danach wird das cwnd auf die Größe der Threshold zurückgesetzt (d.h. wir verbleiben in der Congestion Avoidance-Phase, *fast recovery*).

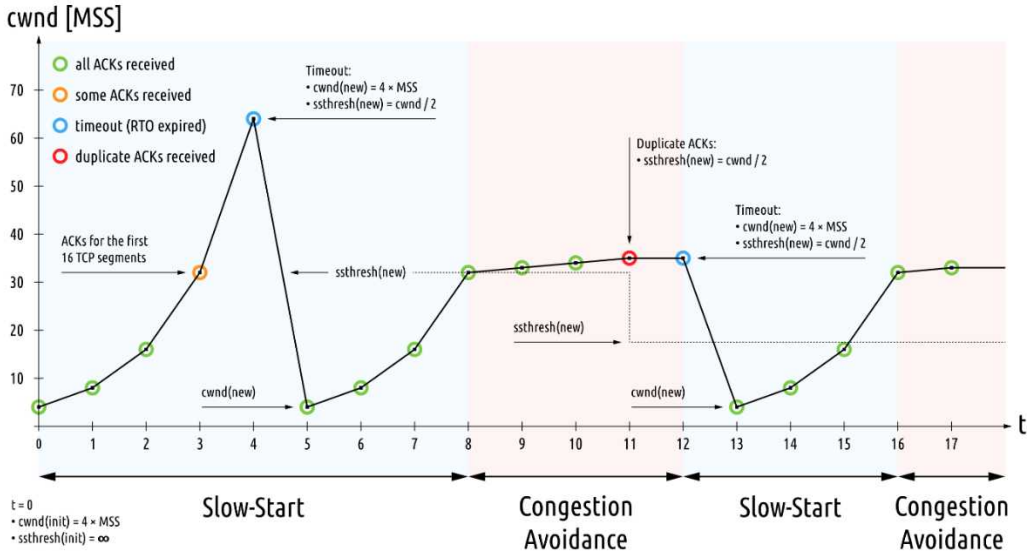


Abbildung 8: Slow-Start/Congestion Avoidance

2 Additive Increase Multiple Decrease:

- Lineare Steigerung der Congestion Window Size bis zum ersten Time Out (Paketverlust)
  - 1,2,3,4,5,6,7,8,9,10
- Sobald ein Time Out festgestellt wird, wird wieder die neue Congestion Window Size berechnet und auf diese zurückgeschaltet
  - new_cwnd = act_cwnd/2 new_cwnd = 10/2 = 5
  - cwnd = 5
- Danach wird wieder linear gesteigert
  - 5,6,7,8,9,10,11,12…

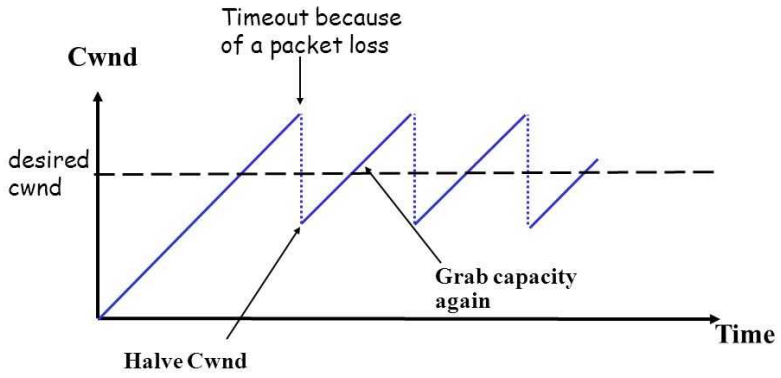


Abbildung 9: Additive Increase Multiple Decrease

6. Vulnerabilities (Schwachstellen)

6.1. SYN-Flooding:

- DoS-Attacke (Denial of Service).
- Es werden solange Anfragen gestellt, bis die maximale Anzahl an möglichen Verbindungen erreicht wurde und der Empfänger nicht mehr antworten kann.
- Dabei wird der Threeway-Handshake missbraucht

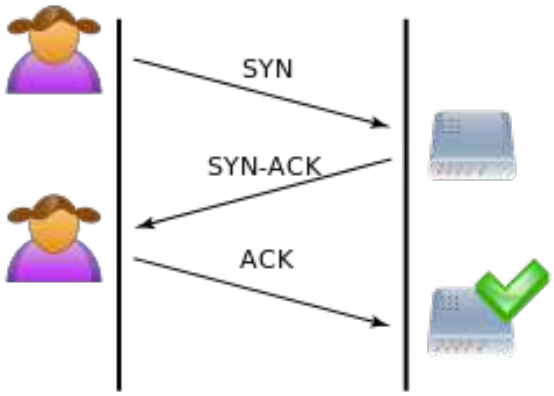


Abbildung 10: Regulärer Threeway-Handshake

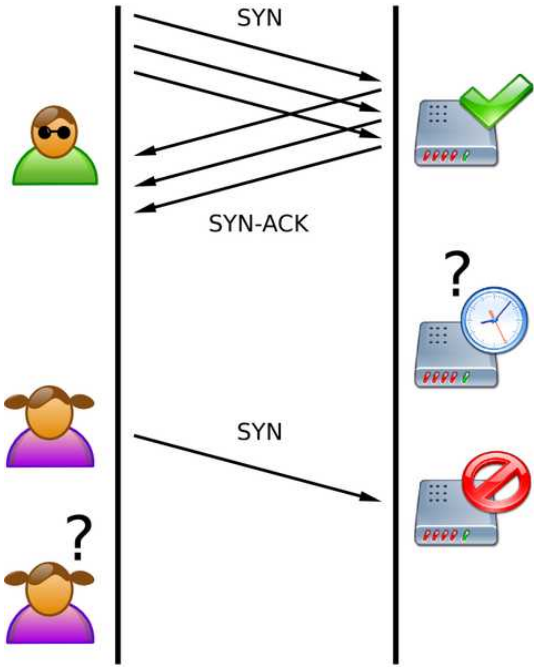


Abbildung 11: Missbrauchter Threeway-Handshake

- Ein böswilliger Client kann die letzte *ACK*-Nachricht unterschlagen
- Der Server wartet einige Zeit auf ein entsprechendes Paket, da es auch aufgrund von Verzögerungen verspätet eintreffen könnte.
    - Adresse und Status des Clients müssen im Speicher gehalten werden
    - Diese halb offene Verbindung belegt also Ressourcen
    - Ressourcen sind immer begrenzt
- Durch eine Flut solcher SYN-Nachrichten braucht der Server alle Ressourcen auf und kann dann auch keine richtigen Verbindungen mehr aufbauen.

    - DOS (Denial of Service)
- Sehr unausgewogener Angriff:
    - SYN-Pakete sehr klein und auch ohne Rechenaufwand einfach erzeugbar
    - Der Verteidiger benötigt mehr Ressourcen zur Abwehr als der Angreifer für den Angriff

6.1.1. *Gegenmaßnahmen*

- Timeout kleiner machen
- SYN-Queue vergrößern
- Eine Echtzeitanalyse des Angriffs durch eine intelligente Firewall, welche verdächtige Angriffsmuster automatisch erkennt.
- Proxy Server

6.2. Sequence Guessing/TCP Sequence Prediction

- Angriffsmethode in TCP/IP-Netzwerken um dem Opfer einen anderen Absender vorzutäuschen (IP-Spoofing) oder bestehende Verbindungen zu übernehmen (TCP/IP-Hijacking).
- Erste Sequenznummer wird beim Verbindungsaufbau ausgehandelt und dann von den Kommunikationspartnern selbständig weitergezählt
- Pakete mit unerwarteten Sequenznummern werden vom Empfänger verworfen.
- Angreifer muss versuchen, die vom Empfänger erwarteten Sequenznummern zu „erraten“
- Gleichzeitig muss er dafür sorgen, dass seine Datenpakete vor denen des tatsächlichen Senders ankommen (z. B. indem der Sender gleichzeitig mit einer Denial-of-Service-Attacke angegriffen wird).
- Gelingt dies, so verwirft der Empfänger daraufhin die Pakete des tatsächlichen Senders (da die entsprechenden Sequenznummern ja bereits vom Angreifer an den Empfänger gesendet wurden). Ab diesem Zeitpunkt scheinen die Pakete des Angreifers für den Empfänger vom erwarteten Sender zu kommen.
- Gelingt es dem Angreifer zusätzlich noch, den tatsächlichen Sender so lange zu blockieren, dass die von ihm mitgezählten Sequenznummern des Empfängers sich um mehr als die Puffergröße von jenen des Empfängers unterscheiden kann er auch dem Sender gefälschte Pakete zuschicken und sich so in die Verbindung zwischen den beiden Kommunikationspartnern einklinken (TCP/IP-Hijacking), da die Sequenznummern dann für beide jeweils außerhalb des erwarteten Bereiches liegen.

6.2.1. *Session Hijacking*

- Entführung einer Kommunikationssitzung
- Ziel des Angreifers ist es, durch die „Entführung“ dieser Sitzung die Vertrauensstellung auszunutzen, um dieselben Privilegien wie der rechtmäßig authentifizierte Benutzer zu erlangen.
    - Der rechtmäßige Benutzer baut eine TCP-Verbindung mittels Drei-Wege-Handschlag auf.
    - Der Angreifer versucht nach erfolgter Authentifizierung den Dialog zu übernehmen, indem er die Antwort-Pakete manipuliert und schneller sendet, als der ursprünglich angesprochene Server oder Client.
    - Dafür muss der Angreifer die Sequenznummer kennen, welche bei unverschlüsselten Verbindungen im Klartext übertragen wird. (Sequence Guessing)

- Session Hijacking ähnelt dem Spoofing-Angriff, allerdings stehen dem Angreifer zu dem Zeitpunkt schon alle notwendigen Informationen zur Verfügung.

6.2.2. *Gegenmaßnahmen*

- Ausschnüffeln der notwendigen Informationen durch verschlüsselte Übertragungen unterbinden.

7. Vergleich UDP und TCP

7.1. Vorteile von UDP

- Geringer Ressourcen Verbrauch
- Kein explizierter Verbindungsaufbau
- Einfache Implementierung
- Multi/Broadcastfähig
- UDP für einfache Client-Server Interaktion geeignet
    - Ein Anfragepaket vom Client
    - Ein Antwortpaket vom Server

7.2. Nachteile von UDP

- Keine Dienstqualität (beliebig hohe Fehlerrate)
- Keine Flusskontrolle (schneller Sender kann langsamen Empfänger überfordern)
- Kein Staukontrollmechanismus (Überlastung im Netz führt zu hohen Verlusten)

7.3. Vorteile von TCP

- Gesicherte Datenübertragung
- Effiziente Datenübertragung trotz Komplexität
- Für geringe Datenraten (z.B. interaktives Terminal) und hohe Datenraten (z.B. Dateitransfer) verwendbar

7.4. Nachteile von TCP

- Höherer Ressourcenbedarf (Cache, Zusatzinfos)
- Verbindungsauf- und –Abbau auch bei kurzen Datenübertragungen notwendig
- Kein Multi/Broadcast möglich

7.5. Eigenschaften von TCP und UPD

Eigenschaft	TCP	UDP
Größe Header	20-60 Byte	8 Byte
TPDU	Segment	Datagram
Checksum	Ja	Optional
Größe der Checksum (Bit)	16	16
Connection-oriented	ja	nein
Full Duplex	ja	ja
Vertrauenswürdige Datenübertragung	ja	nein
Geordnete Nachrichtenübermittlung	ja	nein
Flusskontrolle (Flow control)	ja	nein
Staukontrolle (Congestion control)	ja	nein
ECN (Explicit Congestion	ja	nein

Notification)		
Path MTU Discovery (Dynamische Bestimmung der maximalen Paketgröße)	ja	nein
Automatische Fragmentierung der gesendeten Nachricht	ja	ja (innerhalb von 64K)
Zusammenbauen der einzelnen Teile einer Nachricht	ja	ja (innerhalb von 64K)
Halboffene Verbindungen	ja	nicht anwendbar

7.6. Anwendungen von TCP und UDP

Anwendung	Anwendungsprotokoll	Transportprotokoll
E-Mail	SMTP/IMAP/POP	TCP
Remote-Terminal-Zugang	Telnet/SSH	TCP
Dateiübertragung	FTP/CIFS	TCP
Web	http	TCP (ab Version /3 UDP)
Streaming Audio/Video (VoIP)	RTSP/RTP	TCP (Befehle) + UDP (Übertragung)
Netzwerkverwaltung	SNMP	UDP
Namensauflösung	DNS	UDP