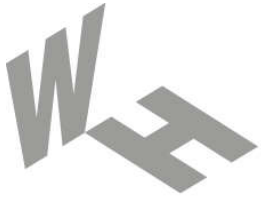


Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Die Transportebene

Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- Zusammenfassung

Inhalt

- **Ziele und Einordnung**
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- Zusammenfassung

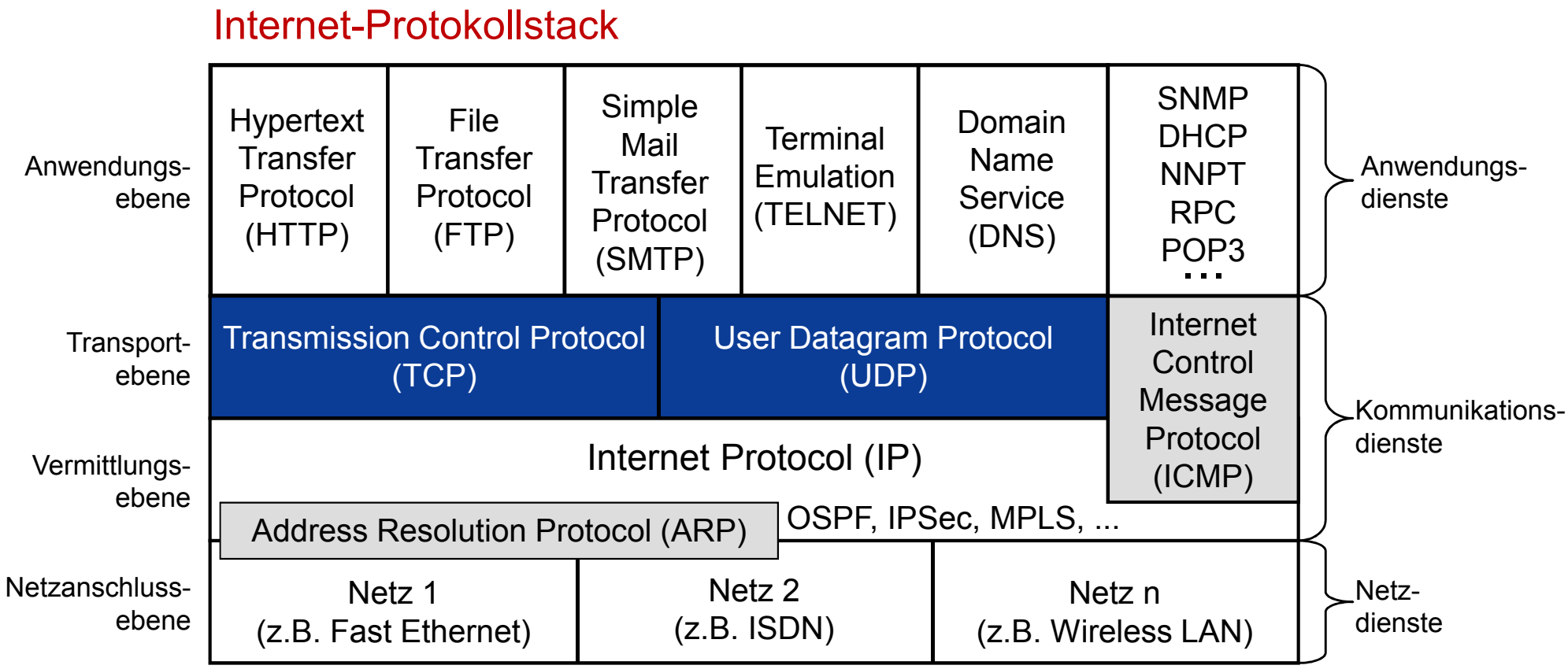
Transportebene
→ Ziele

- Gutes Verständnis für die Transportebene in der TCP/IP-Kommunikationsarchitektur
- Erlangen der Kenntnisse über die Aufgaben, Prinzipien und Mechanismen auf der Transportebene
- Gewinnen von praktischen Erfahrungen über die Transportebene mit Hilfe von Protokollanalysen und Statistiken (IAS)

Transportebene
→ Referenzmodell

- Die ***Transportschicht*** stellt einen netzunabängigen Transportdienst zwischen zwei Endsystemen (end-to-end) bereit.
- Sie bildet die verschiedenen Netzdienste der Vermittlungsschicht mittels geeigneter Transportprotokolle auf den Transportdienst ab.
- **Transportprotokolle sind Endsystem-orientiert.**
- Sie sorgen für den sicheren und bedarfsgerechten Transport von Nachrichten zwischen Endteilnehmern.
- Bedarfsgerecht bedeutet, dass die überlagerte Schicht die Möglichkeit der Auswahl von Güteparametern z.B. für Durchsatz, Verzögerung, Verfügbarkeit oder Restfehlerrate hat.
- **Multiplexen und Splitten von Teilnehmer- oder Anwenderinstanzen-verbindungen** gehört ebenso zu den Schichtenaufgaben wie die **Flußsteuerung**.

Die Transportsebene
→ TCP und UDP - Einordnung



Inhalt

- Ziele und Einordnung
- **Adressierung auf der Transportebene**
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- Zusammenfassung

Adressierung auf der Transportebene
→ Einleitung (1/4)

- Ein Betriebssystem ist in der Regel multitasking-fähig.
- Dies bedeutet, dass mehrere Programme parallel bzw. quasi-parallel ablaufen können.
- Diese Programme laufen in einem oder mehreren Prozessen bzw. Tasks ab.
- Die Identifikation eines Prozesses erfolgt durch eine eindeutige Prozessnummer, der sogenannten Prozess-ID.
- Diese Prozess-ID wird dynamisch beim Start des Prozesses durch das Betriebssystem erzeugt.
- Es ist möglich, dass ein Prozess bei jedem Neustart eine andere Prozess-ID besitzen kann.
- Für die Identifizierung des Prozesses als Ziel einer Nachricht ist die Prozess-ID denkbar ungünstig, da alle potentiellen Sender bei einem Neustart des Zielprozesses darüber informiert werden müssen, wie die neue Prozess-ID lautet.

Adressierung auf der Transportebene
→ Einleitung (2/4)

- Aus diesem Grund wurde eine vom Betriebssystem unabhängige Einheit geschaffen, der **Protokoll-Port** (der Service Access Point (SAP) auf der Transportebene (TSAP) - OSI-Terminologie).
- Das Betriebssystem hat dafür zu sorgen, dass Prozesse zum Senden von Daten eine Portnummer benutzen und leiten beim Empfang von Daten diese dem entsprechenden Prozess zu.
- Das IP-Protokoll regelt den Datenaustausch zwischen den Knoten des IP-Netzes.
- **Mit Hilfe der IP-Adresse können Rechner adressiert werden.**
- Da auf einem Rechner mehrere Prozesse, d.h. mehrere Programme ablaufen können, muss eine zusätzliche Differenzierung, der **Protokoll-Port**, eingeführt werden.
- **Mit Hilfe der Protokoll-Ports, der Transportadresse, können Kommunikationsanwendungen adressiert werden.**
- Die meisten Betriebssysteme stellen einen synchronen Zugriff auf die Ports zur Verfügung.

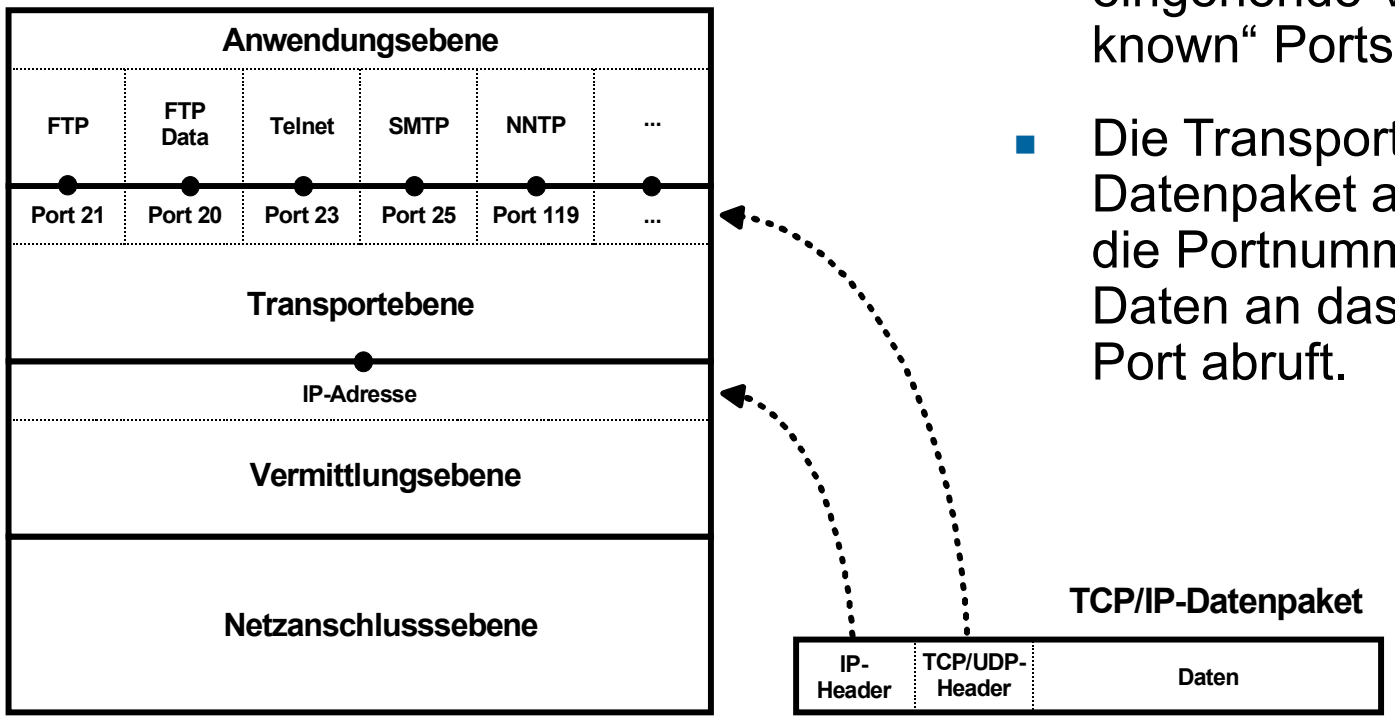
Adressierung auf der Transportebene
→ Einleitung (3/4)

- Synchron bedeutet, dass ein Prozess z.B. bei einem Lesezugriff auf einen Port so lange geblockt wird, bis Daten für diesen Port empfangen werden.
- Sobald Daten empfangen werden, wird der Prozess wieder gestartet.
- Im allgemeinen sind Protokoll-Ports auch gepuffert, so dass Daten zwischengespeichert werden, wenn ein Prozess noch nicht lesebereit ist.

- Eine Applikation stellt den Transport-Dienst an den sogenannten **Sockets** zur Verfügung.
  - Erzeugen des Sockets: SOCKET
  - Binden, Verbindungsauf/abbau: BIND, LISTEN, ACCEPT CONNECT, CLOSE
  - Übertragen: SEND, RECEIVE
- Ein SOCKET kann für mehrere Verbindungen gleichzeitig benutzt werden, d.h. mehrere Verbindungen enden auf dem gleichem SOCKET.

Adressierung auf der Transportebene
→ Einleitung (4/4)

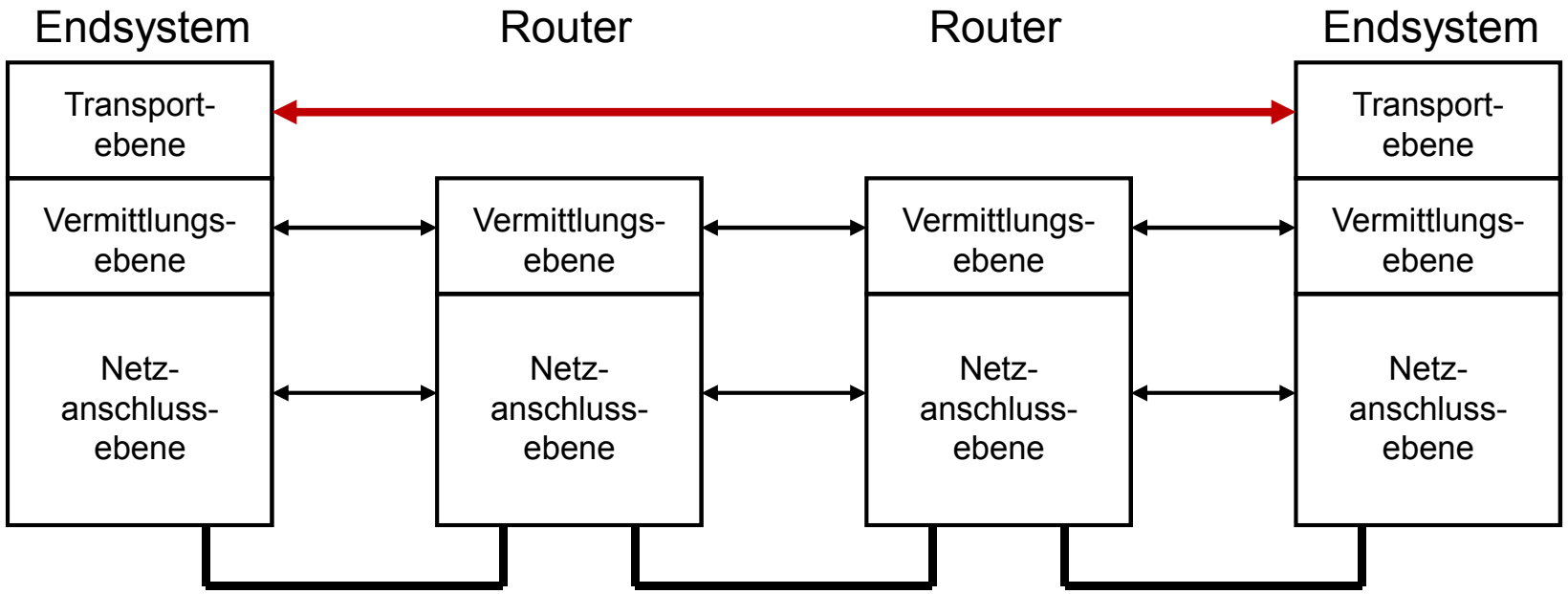
- Man unterscheidet zwischen „well-known“ Ports, die einem Dienst (z.B. HTTP oder SMTP) fest zugeordnet sind, und dynamische Ports, die variabel zugeordnet und bei jedem Nachrichtenaustausch neu vergeben werden.



- Ausgehende Verbindungen werden an freie Ports dynamisch angebunden; wohingegen eingehende Verbindungen „well-known“ Ports ansprechen.
- Die Transportebene nimmt das Datenpaket an, liest aus dem Header die Portnummer und übergibt die Daten an das Programm, das diesen Port abruft.

Adressierung auf der Transportebene
→ Ende-zu-Ende Protokoll

- TCP und UDP sind Ende-zu-Ende Protokolle.
- Nur die miteinander kommunizierenden Endsysteme führen ein Transport-Protokoll aus, nicht die unterwegs befindlichen Transitsysteme (Router).



Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- **UDP – User Datagram Protocol**
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- Zusammenfassung

UDP - User Datagram Protocol
→ Standards

RFC 768

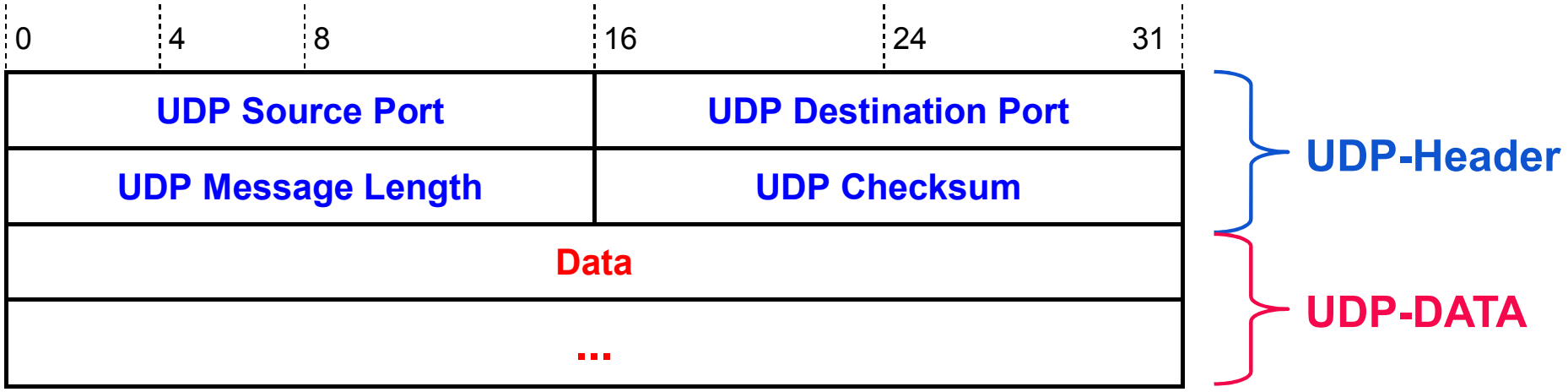
UDP - User Datagram Protocol
→ Einführung (1/2)

- Das User Datagramm Protocol (UDP) ist ein **verbindungsloses** Kommunikationsprotokoll der Transportebene.
- Das User Datagramm Protocol ermöglicht einem Programm das Senden und Empfangen von Datagrammen zu/von anderen Programmen.
- UDP unterstützt dabei die Eigenschaften der Protokoll-Ports, mit denen verschiedene Programme (Kommunikationsanwendungen) auf einem Rechner unterschieden werden können.
- Eine UDP-Nachricht enthält dabei neben den Daten sowohl den Ziel- als auch den Sender-Port, um einerseits der UDP-Software zu ermöglichen, das Ziel zu finden, anderseits auch dem Empfänger zu ermöglichen, eine Antwort zu schicken.
- UDP benutzt IP, um Nachrichten an andere Rechner zu transportieren.
- **UDP** basiert dabei auf demselben **ungesicherten** und **verbindungslosen** Datagramm-Service wie IP.

UDP - User Datagram Protocol
→ Einführung (2/2)

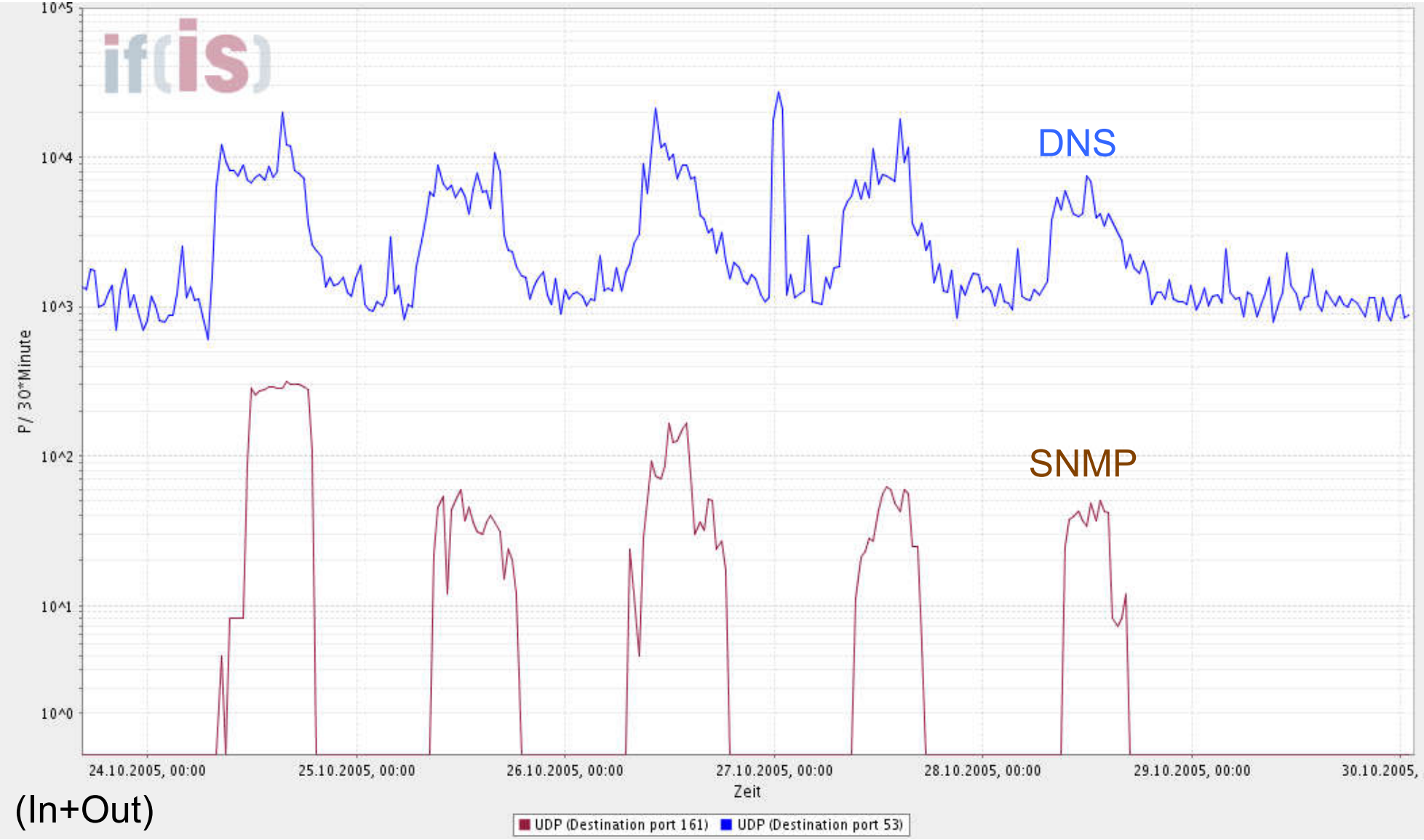
- Es werden keine Quittungen verschickt und eingehende Nachrichten werden auch nicht sortiert.
- Somit können UDP-Nachrichten verlorengehen, gedoppelt werden oder auch defekt ankommen.
- Der entscheidende Vorteil von UDP ist der **geringe Overhead**.
- Dadurch eignet es sich z.B. zur Übertragung von kleinen Datenmengen.
- Hier ist es einfacher, die Daten bei einem auftretenden Fehler einfach nochmal zu übertragen, als eine garantiert fehlerfreie Verbindung aufzubauen.
- Wenn nach einer bestimmten Zeit keine Antwort vom Ziel-Rechner kommt, wird das Datenpaket noch einmal auf den Weg geschickt (Anwendung).
- Eine weitere Möglichkeit ist, einer übergeordneten Anwendung die Sicherheitsfunktion zu überlassen (z.B. SNMP V.3 mit Echtheitsprüfung).
- In diesem Fall ist es unnötig, die Datenübertragung doppelt zu überwachen.

UDP - User Datagram Protocol
→ Das Format einer UDP-Nachricht



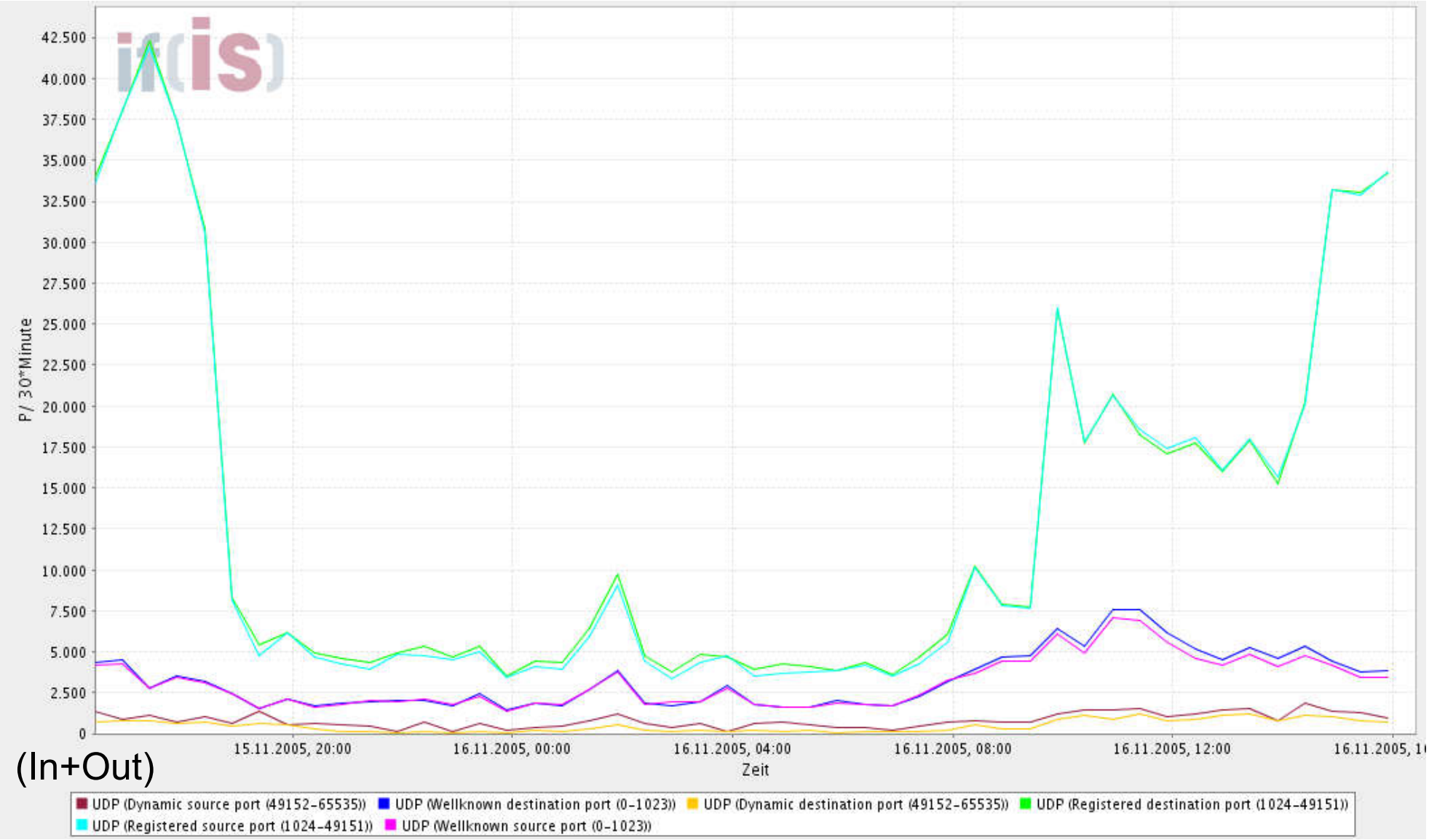
- **Source Port, Destination Port**
  - Feldlänge: 16 Bit → 2^{16} = 65.536 (Ports)
  - Beschreibung
    - Source- und Destination-Port sind 16-Bit UDP Ports, wobei der Source-Port optional ist.
    - Wenn er benutzt wird, so ist dies der Port, an den Antworten geschickt werden können, ansonsten sollte der Wert 0 sein.
- **Length**
  - Feldlänge: 16 Bit
  - Beschreibung
    - Anzahl der Octets im UDP-Datagramm - UDP-Daten und UDP-Header. Der Minimalwert ist 8 - die Länge des Headers ohne Daten.

IAS: FB Informatik
→ Destination Port (UDP)

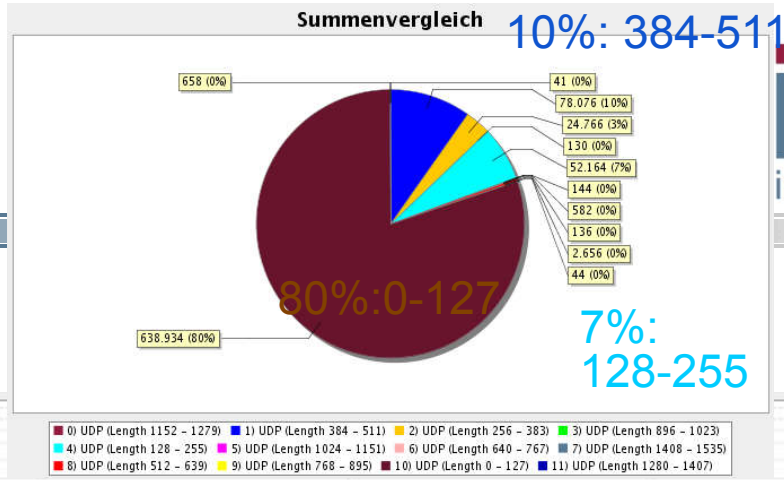


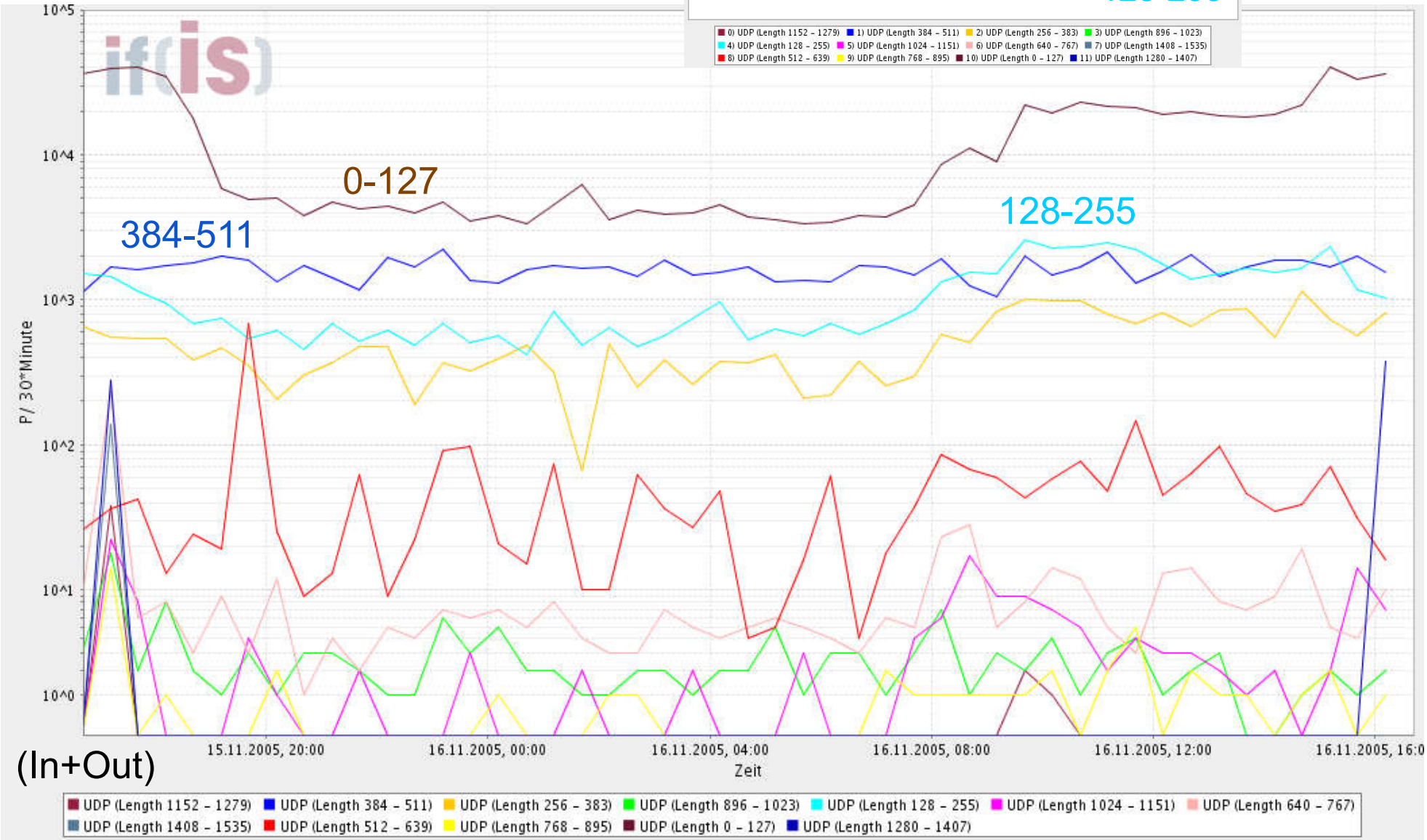
(In+Out)

IAS: FB Informatik
→ Portverteilung nach Portgruppen

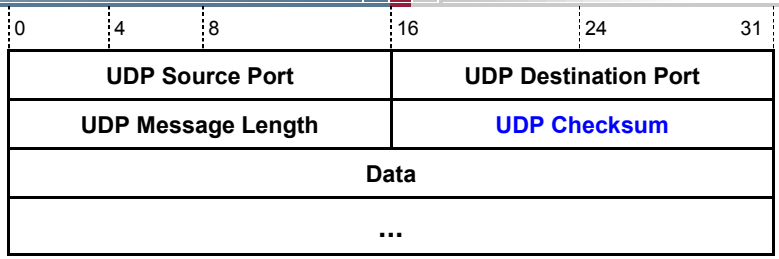


IAS: FB Informatik
→ Length

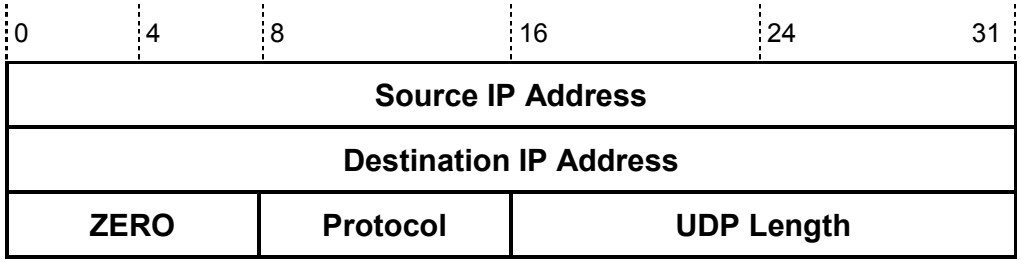




UDP - User Datagram Protocol
→ Das Format einer UDP-Nachricht



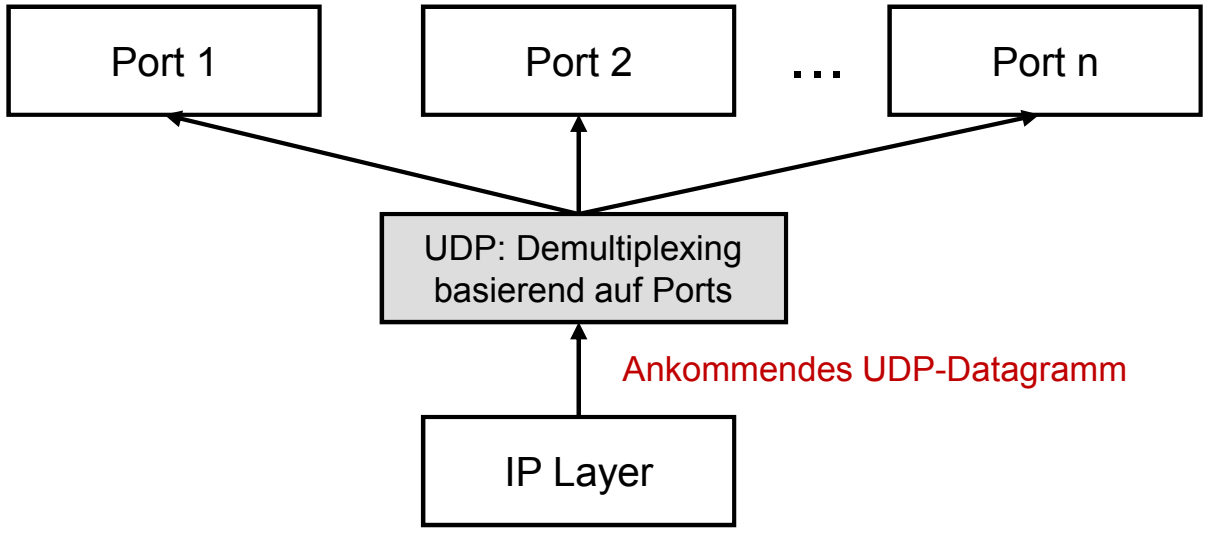
- **UDP Checksum**
  - Feldlänge: 16 Bit
  - **Beschreibung**
    - Die Prüfsumme ist optional. Der Wert 0 bedeutet, dass die Prüfsumme nicht berechnet wurde. **Die Prüfsumme sollte aber immer berechnet werden, es sei denn, die Qualität der Daten spielt keine Rolle (z.B. digitalisierte Sprache)!**
    - Die Prüfsumme beinhaltet neben dem UDP-Header und Daten zusätzlich einen Pseudo-Header.



    - Die Prüfsumme wird gebildet, indem zuerst die Prüfsumme im Datagramm auf 0 gesetzt und dann die 16-Bit Einerkomplement-Summe über Pseudo-Header, UDP-Header und UDP-Daten gebildet wird **(Datenunversehrtheit!)**.
    - Der Sinn des Pseudo-Headers ist die Kontrolle, ob das UDP-Datagramm auch das korrekte Ziel erreicht hat.
    - Das Feld Protocol beinhaltet den Protokoll-Typ (17 für UDP) und das Feld UDP-Length beinhaltet die Länge des UDP-Datagramms ohne Pseudo-Header.

UDP - User Datagram Protocol
→ Verteilung der UDP-Nachrichten auf die Ports

- Die UDP-Software muss ein Multiplexing bzw. Demultiplexing der UDP-Datagramme leisten.
- Erstens muss die UDP-Software von verschiedenen Programmen (Kommunikationsanwendungen) UDP-Datagramme entgegennehmen und diese an die IP-Software weiterleiten.
- Zweitens müssen die UDP-Datagramme, die von der IP-Software an die UDP-Software gegeben werden, abhängig von der Portnummer an die entsprechenden Kommunikationsanwendungen verteilt werden.



UDP - User Datagram Protocol
→ Verteilung der UDP-Nachrichten auf die Ports

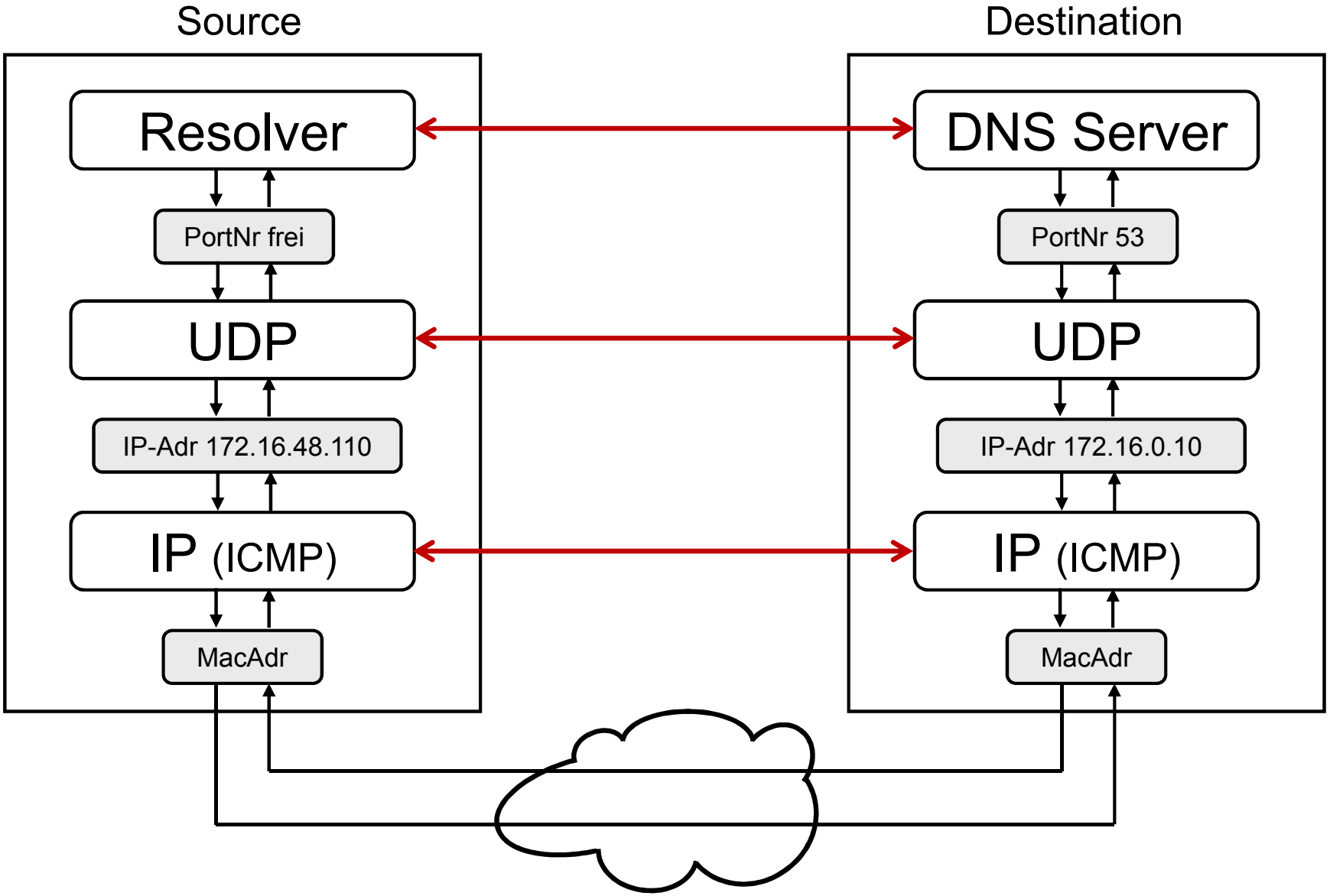
- In den meisten Implementierungen stellt das Betriebssystem einer Applikation, die einen Protokoll-Port benutzt, eine Queue zur Verfügung.
- In dieser Warteschlange schreibt die UDP-Software die Nachricht für die Applikation, wenn ein UDP-Datagramm mit der entsprechenden Port-Nummer empfangen wird.
- Somit ist eine Zwischenspeicherung der Daten gewährleistet, wenn die Applikation einige Zeit lang keine Daten entgegen nehmen kann.

UDP - User Datagram Protocol
→ Reserviert Portnummern („well-known"-Ports)

- Eine Applikation muss, um Daten über UDP an eine andere Applikation verschicken zu können, die Portnummer dieser Applikation kennen.
- Für UDP sind einige Portnummern weltweit eindeutig reserviert.
- Des weiteren stehen eine große Anzahl von Portnummern für lokale Applikationen zur Verfügung.

Portnummer	Bezeichnung	Beschreibung
53	DNS	Domain Name Service
69	TFTP	Trivial File Transfer Protocol
161	SNMP	Simple Network Management Protocol
5060	SIP	Session Initiation Protocol

UDP - User Datagram Protocol
→ Beispiel: DNS Anfrage



UDP - User Datagram Protocol
→ Beispiel: DNS Anfrage

Log einer UDP Verbindung (Übersicht)

```
No.  Time       Source          Destination     Protocol Info
  1  0.000000   172.16.48.110   172.16.0.10     DNS      Standard query A www.heise.de
  2  0.007927   172.16.0.10     172.16.48.110   DNS      Standard query response A 193.99.144.71
```

Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- **TCP - Transmission Control Protocol**
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- Zusammenfassung

TCP - Transmission Control Protocol
→ Standards

RFC 793

TCP - Transmission Control Protocol
→ Einleitung (1/2)

- Das Transmission Control Protocol (TCP) ist ein **verbindungsorientiertes** Kommunikationsprotokoll der Transportebene, welches eine **gesicherte Datenübertragung garantiert** und auf dem **Stream-Mechanismus** basiert.
- Verteilte Applikationen müssen oft sehr große Datenmengen zwischen den Rechnern austauschen.
- Eine solche Applikation auf der Basis eines ungesicherten, verbindungslosen Protokolls wie z.B. UDP würde bedeuten, dass in **jede Applikation** Mechanismen zur Fehlererkennung und zur Wiederholung der Daten eingebaut sein müssten.
- Um dies zu vermeiden, wurden diese Funktionalitäten in die Transportschicht verlagert und ein gesichertes, stream-orientiertes Protokoll - das Transmission Control Protocol - entwickelt.
- Der **Stream-Service** hat dabei die Aufgabe, dem Empfänger die Daten in genau derselben Reihenfolge zukommen zu lassen, wie diese vom Sender verschickt werden.

TCP - Transmission Control Protocol
→ Einleitung (2/2)

- Das Transmission Control Protocol (TCP) wurde spezifisch zur Bereitstellung eines **zuverlässigen Bytestroms von Ende-zu-Ende** in einem **unzuverlässigen Netzverbund** entwickelt.
- Ein Netzverbund unterscheidet sich von einem Einzelnetz dahingehend, dass verschiedene Teile eventuell total unterschiedliche Topologien, Bandbreiten, Verzögerungen, Paketgrößen und andere Parameter haben.
- TCP wurde entwickelt, um die verschiedenen Merkmale von Verbundnetzen **dynamisch anzupassen** und das gesamte **„Kommunikationssystem“ robuster zu machen**!

TCP - Transmission Control Protocol
→ Virtuelle Verbindungen

- Vor dem Beginn der Kommunikation informiert sowohl der Sender als auch der Empfänger das Betriebssystem, dass eine Verbindung aufgebaut werden soll.
- Einer der Partner sendet eine Anforderung, welche vom anderen Kommunikationspartner akzeptiert wird.
- Nach der Durchführung der Verbindungsaufnahme wird die Applikation des Kommunikationspartners informiert, dass der Datenaustausch beginnen kann.
- Im Fehlerfall, wenn die Verbindung z.B. unterbrochen wird, stellen beide Kommunikationspartner diese fest und melden es an die Applikation.
- Die Verbindung ist aber nur **virtuell**, da das darunter liegende IP-Protokoll weiterhin auf dem **Datagramm-Service** beruht.
- TCP ist verbindungsorientiert, d.h. es gibt einen
  - Verbindungsaufbau,
  - Datentransfer und
  - Verbindungsabbau.

TCP - Transmission Control Protocol
→ Gepufferte Datenübertragung

- Der Protokollsoftware steht es frei, den Datenstrom in Pakete zu unterteilen, unabhängig von den Datenpaketen, welche die Applikation übergibt.
- Wichtig ist nur, dass die Daten wieder in **der selben Reihenfolge** zusammengesetzt werden.
- Sendet eine Applikation z.B. ein Octet nach dem anderen, so kann, um eine höhere Effizienz zu erzielen und die Netzlast zu minimieren, die Protokoll-Software so lange **Daten sammeln**, bis ein genügend großes Datagramm entsteht (**Nagle Algorithmus**).
- Entsprechend können auch extrem **große Datenpakete in kleine Blöcke unterteilt** werden (meist ungefähr 1.500 Byte).
- Für Applikationen, die einen Transfer der Daten auch in kleinen Einheiten benötigen, wurde der **PUSH-Mechanismus** eingeführt, der eine Datenübertragung unmittelbar anregt (siehe PM bei HTTP).

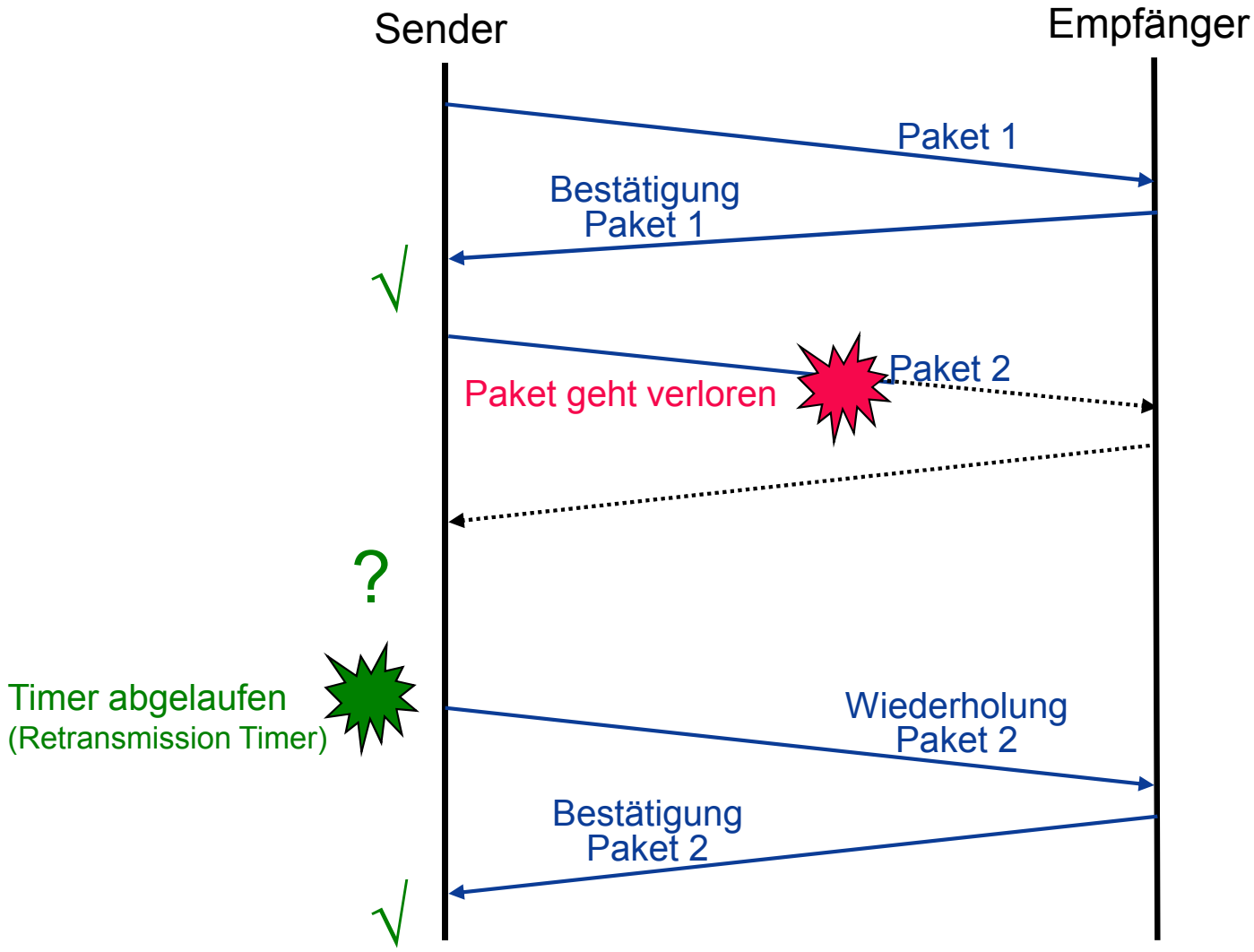
TCP - Transmission Control Protocol
→ Full-Duplex Verbindung

- Verbindungen über TCP ermöglichen eine Full-Duplex Kommunikation, d.h. beide Applikationen können gleichzeitig senden.
- Eine Full-Duplex Verbindung besteht aus **zwei unabhängigen Streams** in gegensätzlicher Richtung.
- Beide Streams können unabhängig voneinander geschlossen werden, was die Verbindung zu einer Half-Duplex Verbindung macht.
- In der Praxis wird aber fast immer, nur eine Full-Duplex Kommunikation durchgeführt (wenn eine Seite abbaut, baut die andere Seite auch ab).

TCP - Transmission Control Protocol
→ Gewährleistung der gesicherten Übertragung

- Um die **Übertragung zu gewährleisten**, wurde beim TCP-Protokoll der Mechansimus **„acknowledgement with retransmission“** gewählt.
- Jedes Paket wird vom Empfänger bestätigt!
- Bleibt eine Bestätigung aus, schickt der Sender das Paket erneut.
- Hierzu muss der Sender das Paket so lange aufbewahren, bis eine Bestätigung vom Empfänger eintrifft.
- Dazu muss der Sender einen Timer (**Retransmission Timer**) starten, um gegebenenfalls ein Paket erneut zu senden, falls innerhalb der eingestellten Zeit keine Bestätigung angekommen ist.
- Bei dieser Vorgehensweise kann es natürlich auch vorkommen, dass ein Paket beim Empfänger doppelt ankommt, wenn z.B. die Bestätigung vom Empfänger verloren geht.
- Deshalb müssen **Sequenznummern** eingeführt werden, damit der Empfänger gegebenenfalls doppelt empfangene Pakete eliminieren kann.
- Die Sequenznummer dient ebenfalls zur korrekten Aneinanderreihung der Daten im Empfänger.

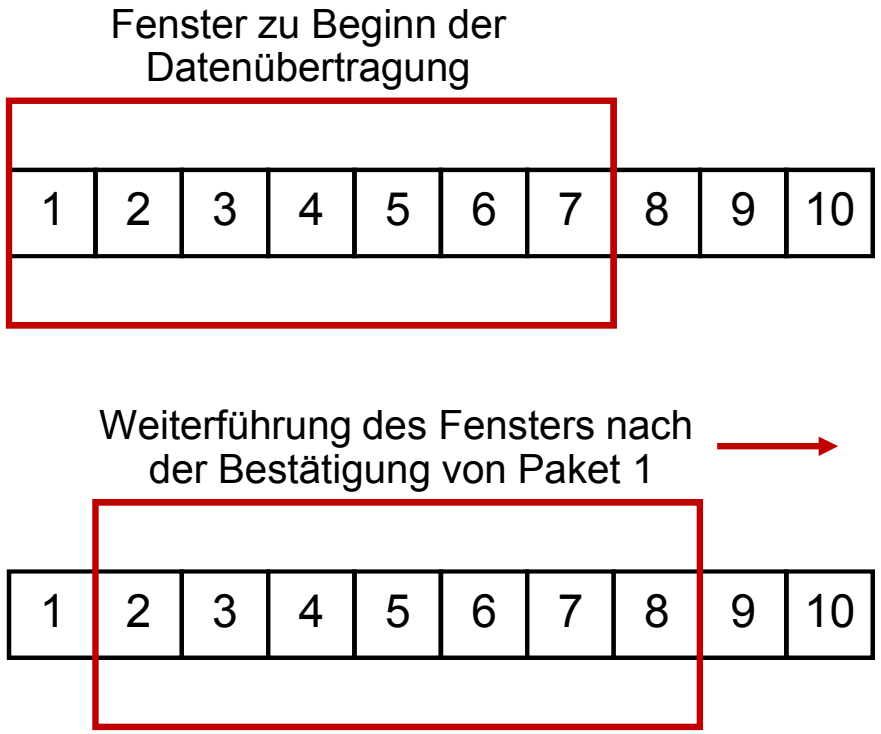
TCP - Transmission Control Protocol
→ Gewährleistung der gesicherten Übertragung



TCP - Transmission Control Protocol
→ Die „Sliding-Window“ Technik

- Der Mechansimus „acknowledgement with retransmission“ erhöht die Übertragungssicherheit enorm, wirkt sich aber in dieser einfachen Form sehr negativ auf die **Bandbreitennutzung** des Netzes aus.
- Der Sender muss nach dem Senden immer warten, bis der Empfänger eine Bestätigung des Paketes geschickt hat.
- Somit wird von der Full Duplex Verbindung zu einer Zeit immer nur ein Kanal benutzt.
- Mit der **Sliding-Windows Technik** (Schiebefenster Technik) wird **die Bandbreite effektiver ausgenutzt**.
- Dem Sender wird dabei ermöglicht, mehrere Pakete zu verschicken, bevor er eine Bestätigung erwartet.
- Stellt man sich die zu transportierenden Daten als eine Sequenz von Paketen vor, so wird zu Beginn der Übertragung ein Fenster einer definierten Länge darüber gelegt.
- Dem Sender ist nur erlaubt, die Anzahl der Pakete zu versenden, die sich innerhalb des Fensters befinden, ohne auf eine Bestätigung zu warten.

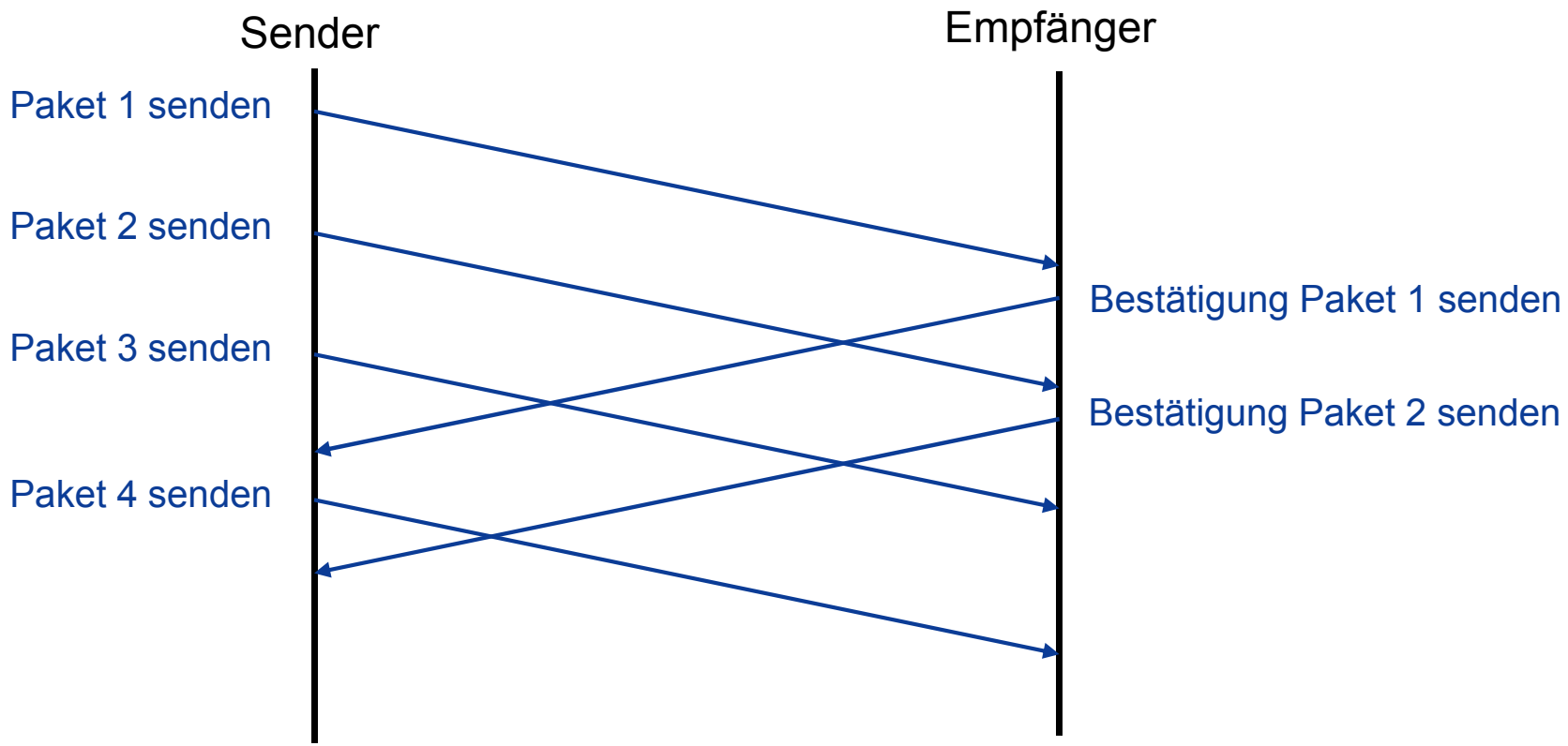
TCP - Transmission Control Protocol
→ Die „Sliding-Window“ Technik



- Das Fenster wird erst dann weiter bewegt, wenn das jeweils erste Paket bestätigt wird.
- Das Protokoll muss sich hierzu jedes nicht bestätigte Paket merken und einen **Timer** für jedes abgeschickte Paket betreiben, um Pakete, die nicht in der vorgegebenen Zeit bestätigt wurden, erneut zu verschicken.

TCP - Transmission Control Protocol
→ Die „Sliding-Window“ Technik

- Beispiel für das Senden von Daten bei einer Windows-Size von 3



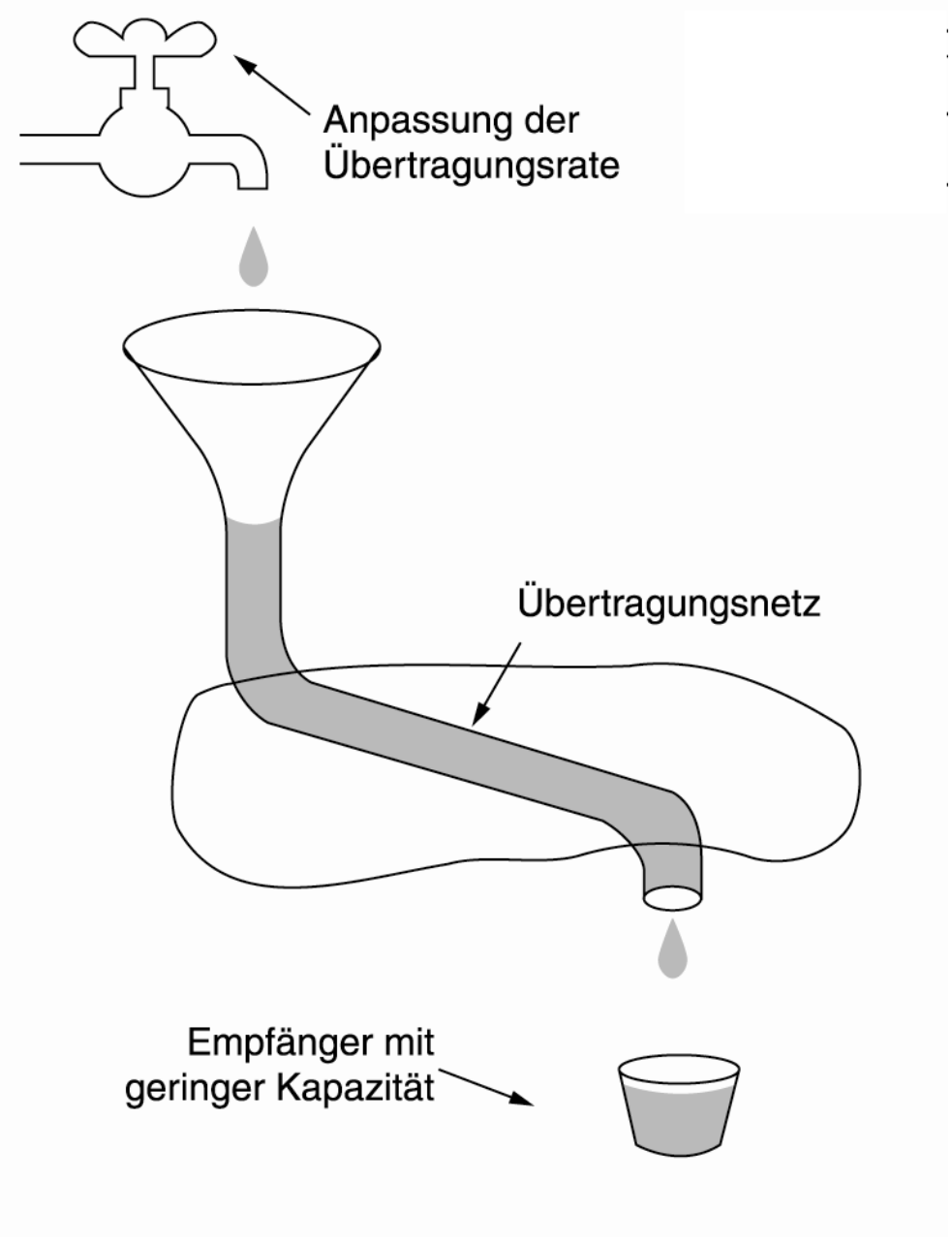
TCP - Transmission Control Protocol
→ Die „Sliding-Window“ Technik

- Bei TCP wird ***nicht*** mit der Einheit „Pakete“ gerechnet.
- Bei TCP wird direkt auf dem Datenstrom (Sequenz) von Octets (Bytes) gearbeitet und ebenso die Fenstergröße in Anzahl Octets angegeben.
- Dies ist sinnvoll, da die Squenznummer bei der Datenübertragung ebenfalls in der Einheit der Octets hochgezählt wird, und somit die Größe der Segmente flexibel bleibt.
- Die TCP-Software im Sender hat drei Zeiger, um den Datenstrom zu verwalten.
- Der **erste Zeiger** markiert die **linke Seite des Fensters**, das die bestätigten von den unbestätigten Octets trennt.
- Ein **zweiter Zeiger** markiert die **rechte Seite des Fensters** und trennt damit die Octets, die ohne Bestätigung noch gesendet werden dürfen von den Octets, die erst bei einer Weiterführung des Fensters an der Reihe sind.
- Der **dritte Zeiger** liegt **innerhalb des Sendefensters** und markiert die Grenze zwischen den gesendeten und noch nicht gesendeten Octets.

TCP - Transmission Control Protocol

→ Analogie: Empfangspufferüberlauf/Flow Control Window

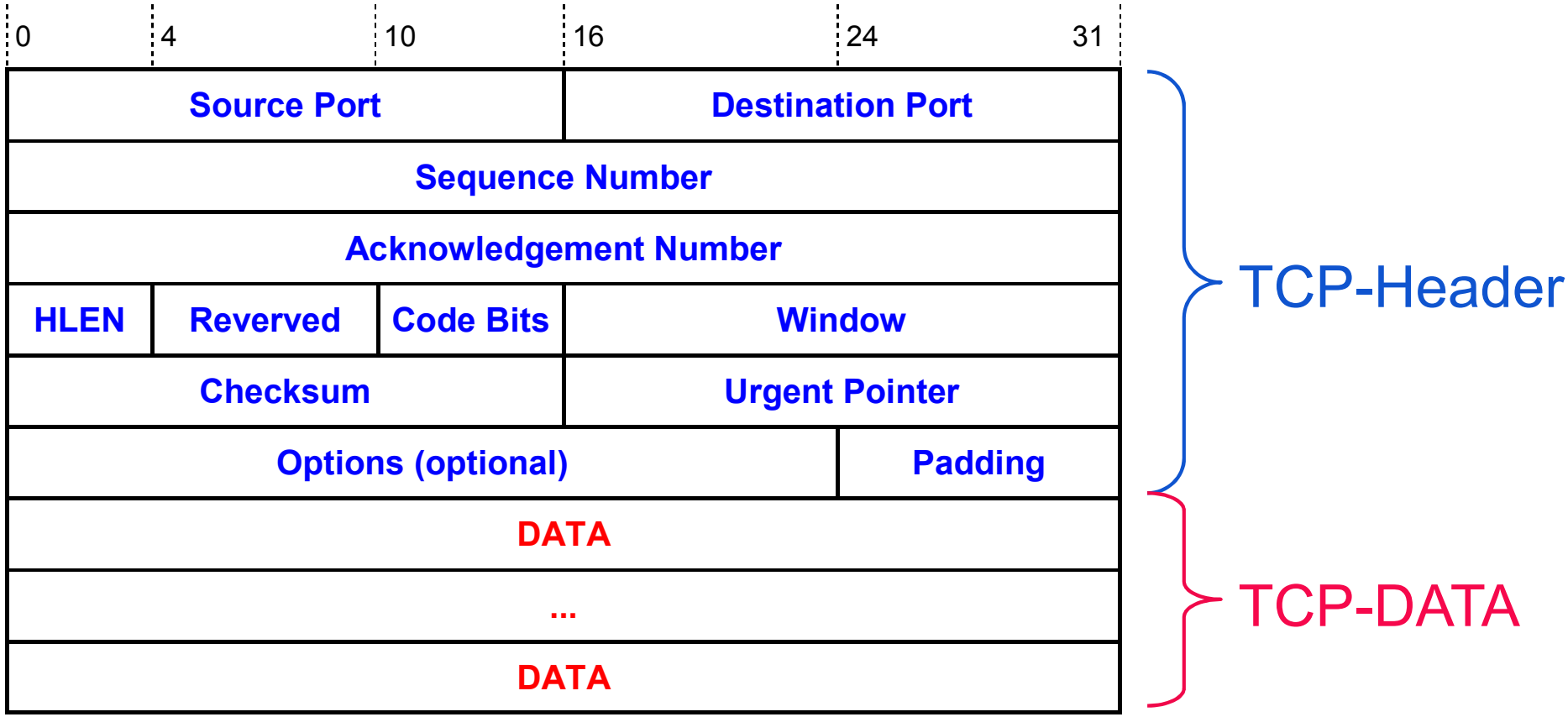
- In dieser Analogie sehen wir ein dickes Rohr, das zu einem Behälter mit geringer Kapazität führt.
- Solange nicht mehr Wasser eingefüllt wird, als der Behälter aufnehmen kann, geht kein Wasser verloren!



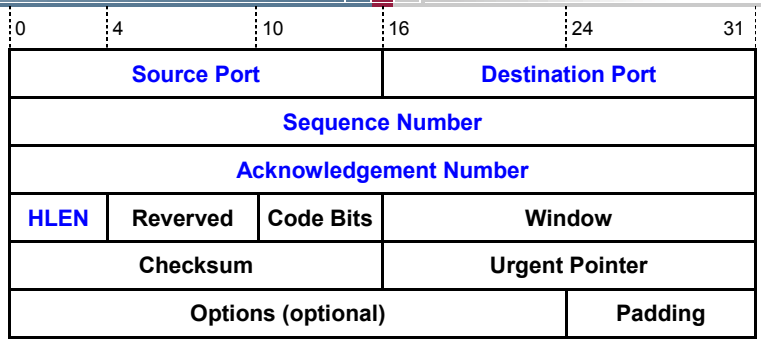
TCP - Transmission Control Protocol
→ Variable Window-Size

- Zusätzlich zu der eingeführten Sliding-Window Technik ist es bei TCP auch noch möglich, die Windows-Size während der Laufzeit zu ändern (Flow Control Window).
- Jede Bestätigung vom Empfänger enthält eine Angabe darüber, wie viele Octets der Empfänger bei weiteren Sendungen zu empfangen bereit ist (Window-Size).
- Dies hat den Vorteil, dass ein Empfänger so z.B. einem **Empfangspufferüberlauf rechtzeitig vorbeugen** kann.
- Sendet der Sender schneller, als der Empfänger die Daten verarbeiten kann, so verringert der Empfänger die Window-Size so weit, dass der Puffer nicht überläuft.
- Im Extremfall kann die Window-Size sogar auf 0 gesetzt werden, was einer Aufforderung zum Stoppen der Datenübertragung gleichkommt.
- Dieser **Mechanismus zur Flusskontrolle** ist entscheidend, da in einem IP-Netz Maschinen unterschiedlicher Größe und Geschwindigkeit vorkommen können.

TCP - Transmission Control Protocol
→ Format

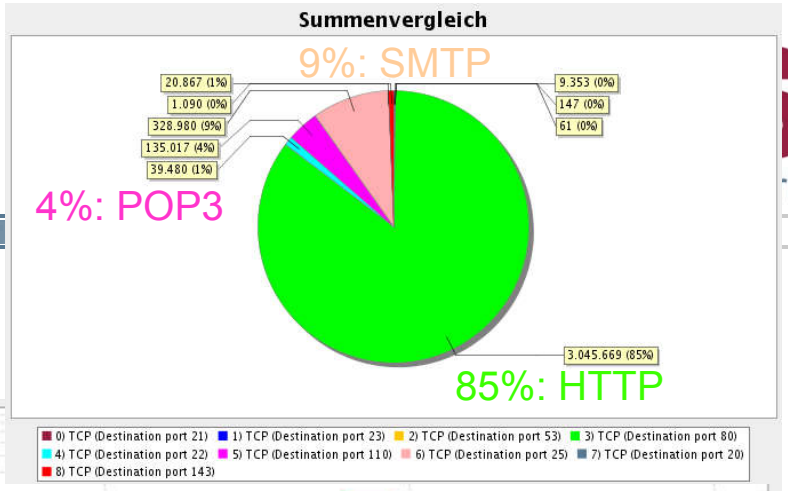


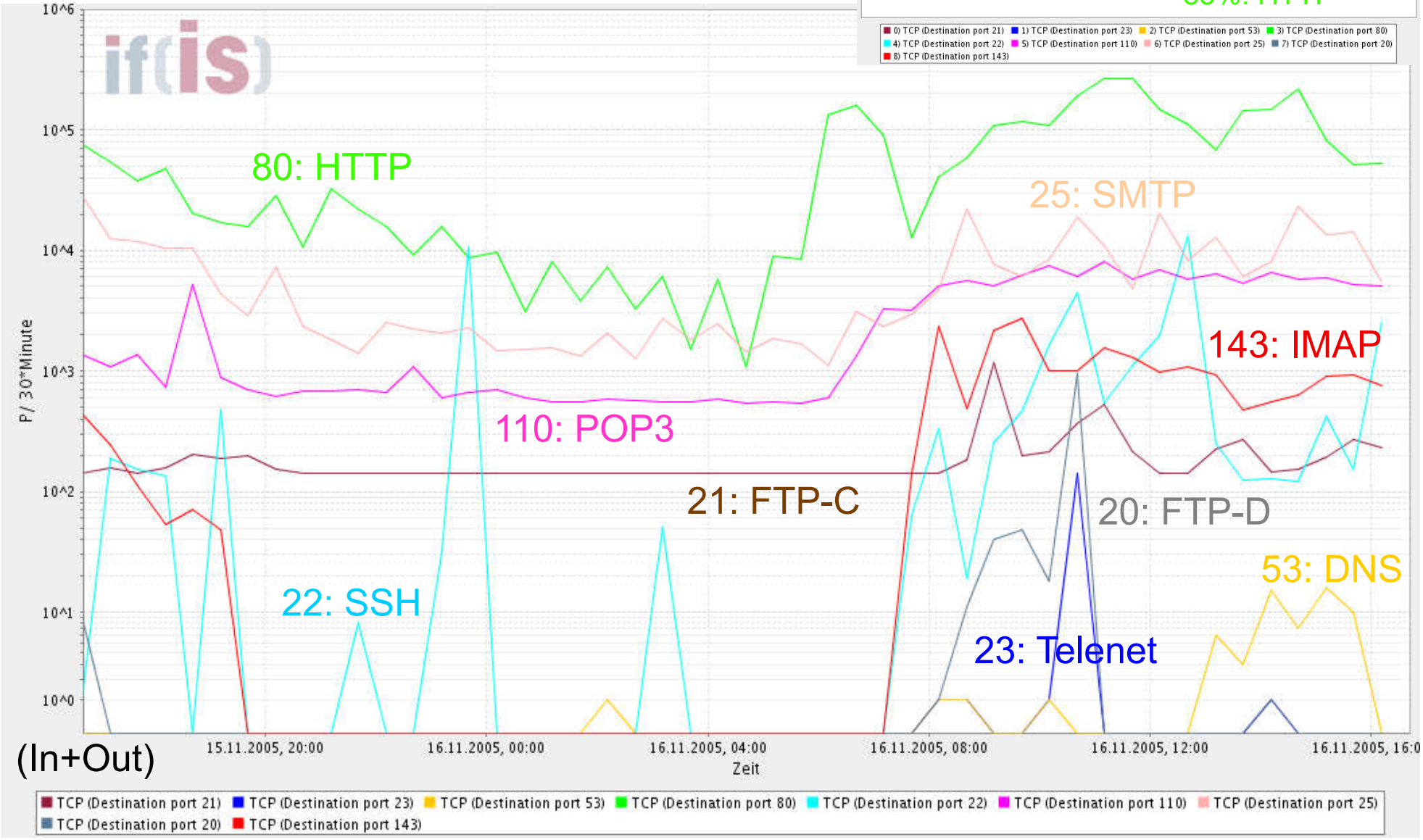
TCP - Transmission Control Protocol
→ Feldelemente des TCP-Headers (1/5)



- **Source- und Destination Ports**
  - Feldlänge: 16 Bit → 2^{16} = 65.536 (Ports)
  - **Beschreibung**
    - Enthalten die Portnummern der Applikationen
- **Sequence Number**
  - Feldlänge: 32
  - **Beschreibung**
    - enthält die Position des Pakets im Octet-Stream
- **Acknowledgement Number**
  - Feldlänge: 32
  - **Beschreibung**
    - enthält die Sequenznummer des Octets, welches der Empfänger als nächstes empfangen will
- **HLEN**
  - Feldlänge: 4
  - **Beschreibung**
    - Die Länge des Headers in 32-Bit Werten (da Options variabel ist)

IAS: FB Informatik
→ Destination Port (TCP)



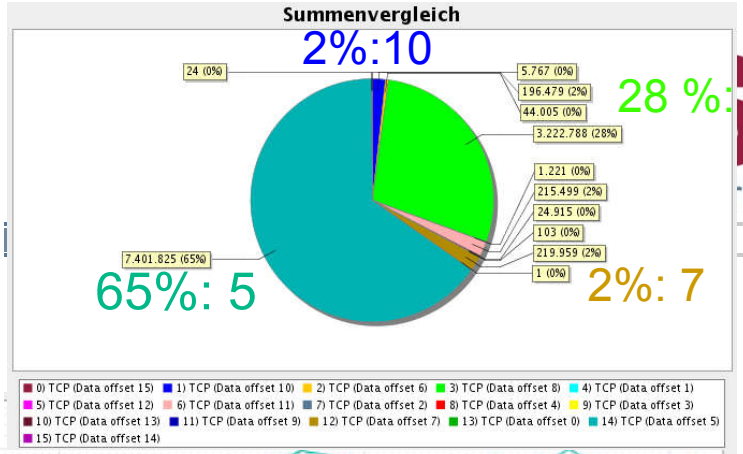


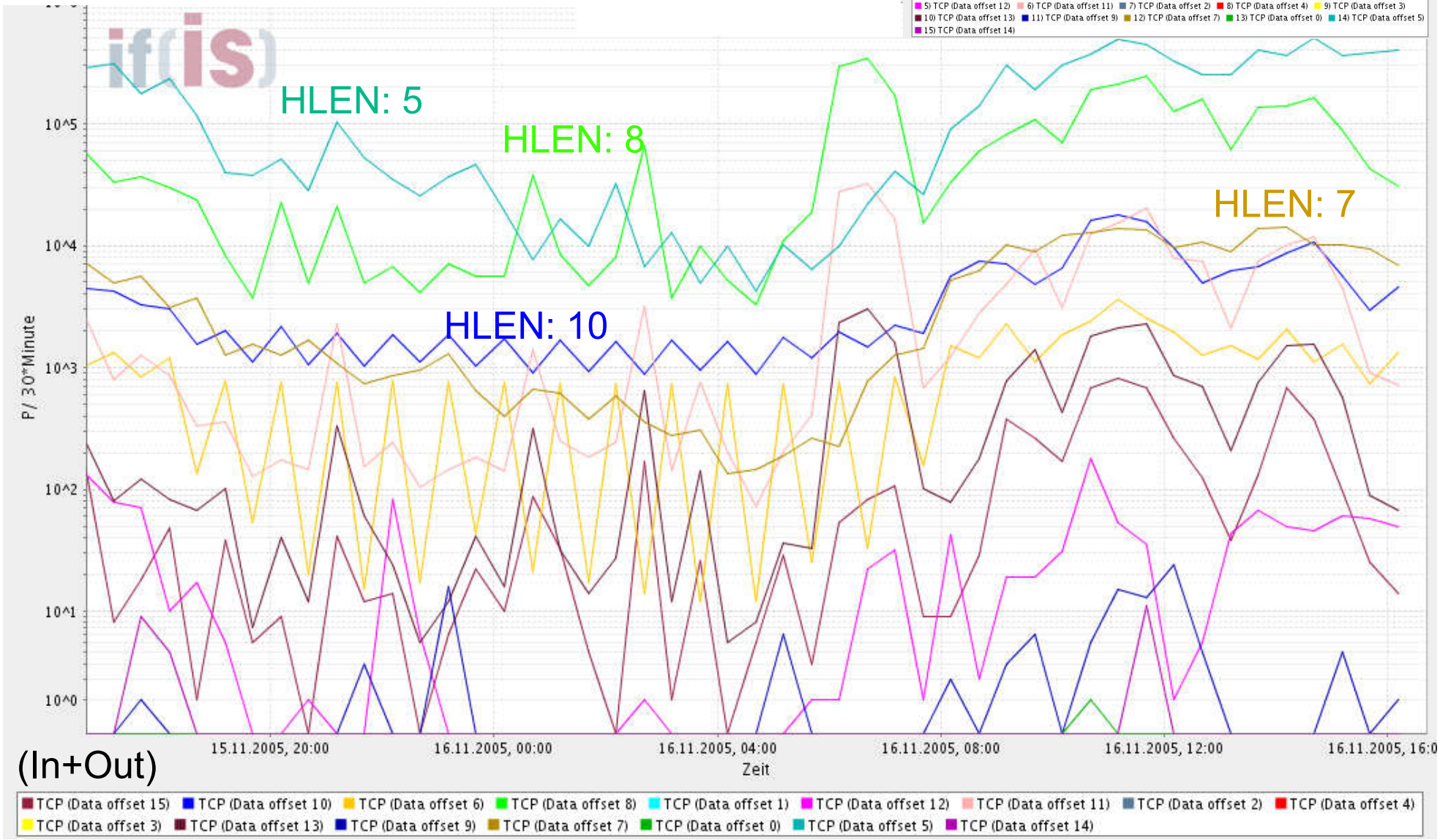
IAS: FB Informatik
→ HLEN

HLEN: 8
NOP (2 Byte)
Timestamp (10 Byte)

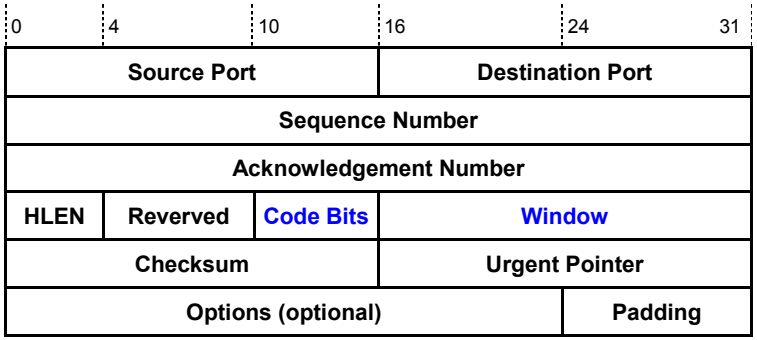
HLEN: 7
MSS (4 Byte)
NOP (2 Byte)
SACK permitted (2 Byte)

HLEN: 10
NOP (2 Byte)
SACK (18 Byte)
oder
MSS (4 Byte)
SACK permitted (2 Byte)
NOP (1 Byte)
WScale (3 Byte)





TCP - Transmission Control Protocol
→ Feldelemente des TCP-Headers (2/5)

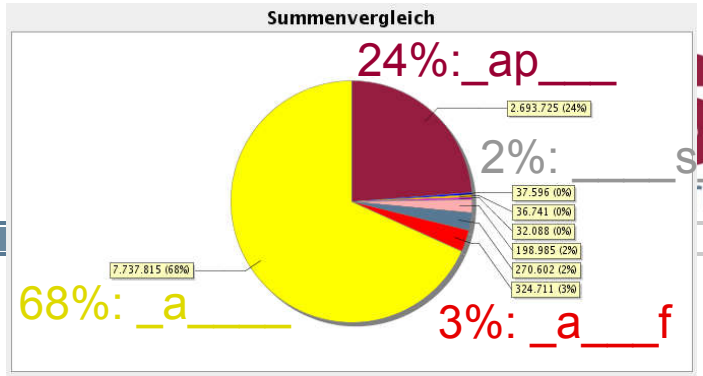


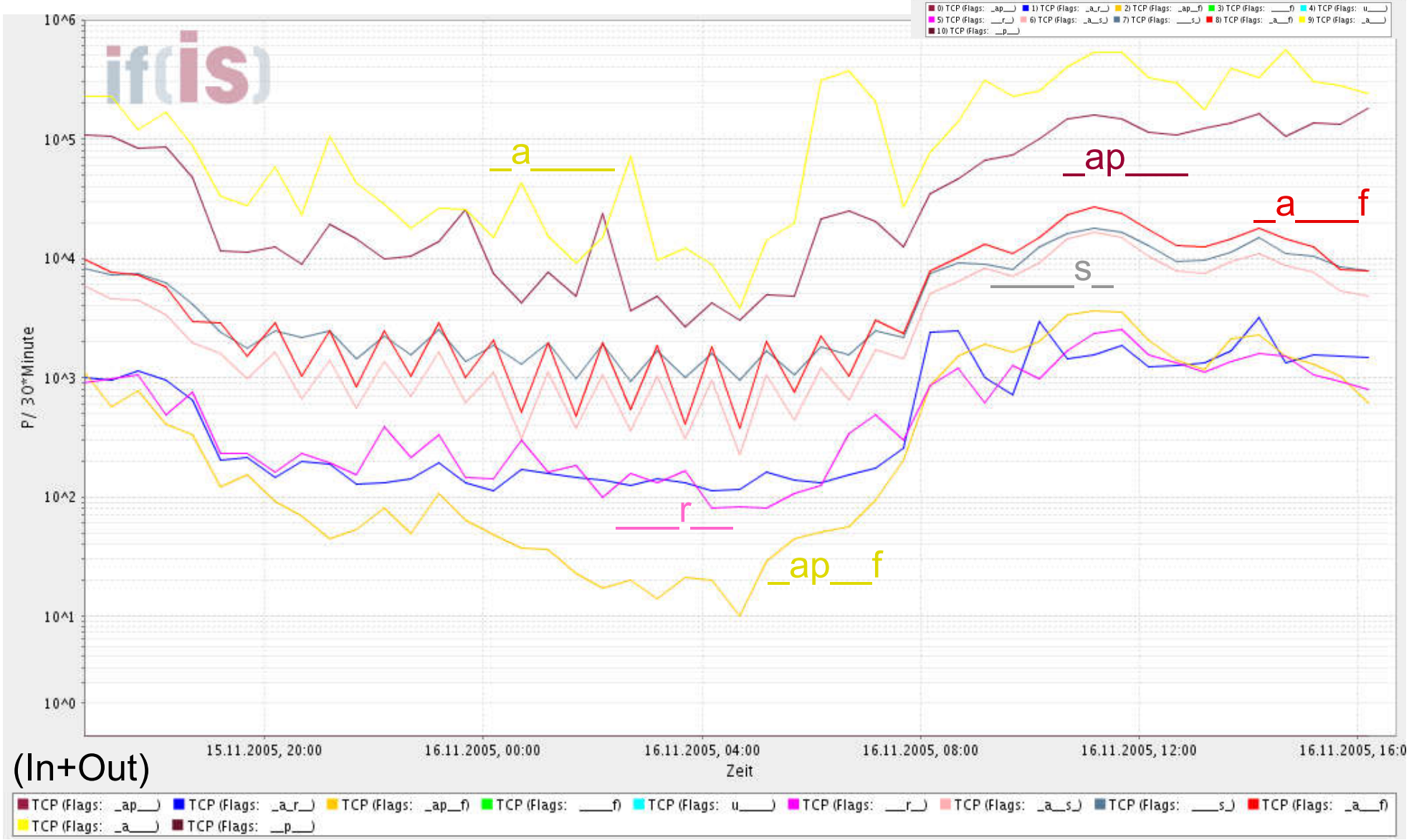
- **Code Bits**
  - Feldlänge: 6 Bit
  - **Beschreibung**

Bits (von links nach rechts)	**Wenn gesetzt (=1)**
URG	Urgent-Pointer ist gültig (Interrupt Nachricht)
ACK	Bestätigung eines Segments (acknowledgement Feld ist gültig).
PSH	(push) Der Empfänger soll die Daten der Anwendung so schnell wie möglich zur Verfügung stellen.
RST	Reset der Verbindung
SYN	Synchronisation der initialen Sequenz-Nummer bei Verbindungsaufbau
FIN	Der Sender hat das Senden seiner Daten beendet (Ende der Übertragung).

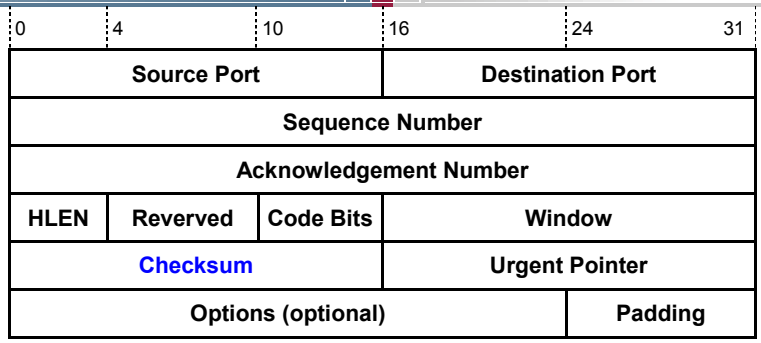
- **Window**
  - Feldlänge: 16
  - **Beschreibung**
    - Angabe der Größe des Empfangsfenster (Einheit in Octets und Vielfaches der Segment-Größe).

IAS: FB Informatik
→ Code Bits

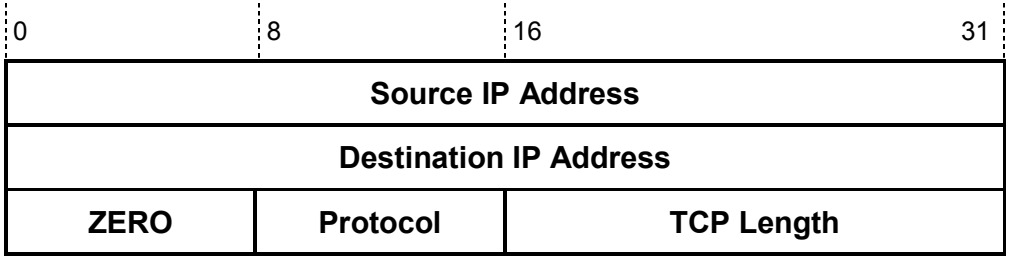




TCP - Transmission Control Protocol
→ Feldelemente des TCP-Headers (3/5)

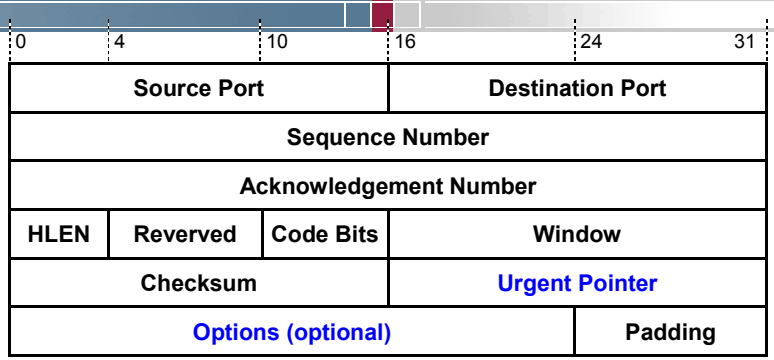


- **TCP Checksum**
  - Feldlänge: 16 Bit
  - **Beschreibung**
    - Die Berechnung der Prüfsumme erfolgt, indem zuerst (wie bei UDP) ein Pseudo-Header vor das TCP-Segment gestellt wird.



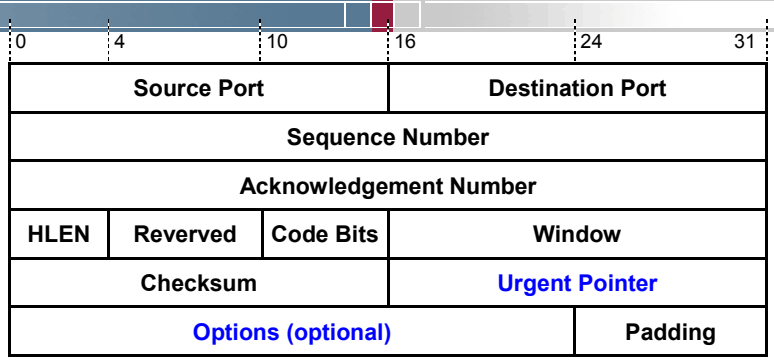
    - Der Pseudo-Header hat den Sinn festzustellen, ob das TCP-Paket den korrekten Empfänger erreicht hat.
    - Das Feld „Protocol“ enthält für TCP den Wert 6.
    - Das Feld „TCP-Length“ ist die Länge des gesamten TCP-Segments inklusive des Headers, aber ohne Pseudo-Header.
    - Berechnet wird die Prüfsumme als 16-Bit Einerkomplement-Summe über Pseudo-Header, TCP-Header und TCP-Daten **(Datenunversehrtheit!)**.

TCP - Transmission Control Protocol
→ Feldelemente des TCP-Headers (4/5)



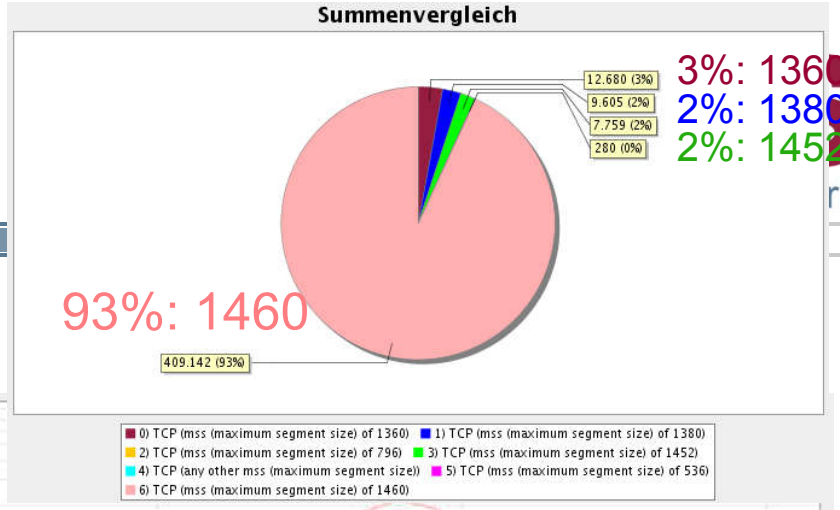
- **Urgent-Pointer**
  - Feldlänge: 16 Bit
  - **Beschreibung**
    - In manchen Fällen ist es notwendig, Out-of-Band Daten zu verschicken, die, selbst wenn im Empfangspuffer des Empfängers noch Daten vorliegen, sofort an die Applikation weitergegeben werden müssen.
    - Dies ist z.B. bei einem Remote Login notwendig, wenn ein Abbruch des Login-Vorgangs erzwungen werden soll (Ctrl-C).
    - Wenn ein Urgent-Pointer von TCP empfangen wird, muss TCP der Empfänger-Applikation den Urgent-Modus mitteilen, genauso, wie eine Rückkehr in den Normal-Modus nach Abschluss des Empfangs der Urgent-Pakete mitgeteilt werden muss. Wenn das URG-Bit gesetzt ist, weist der Urgent-Pointer auf die Position in den Daten innerhalb des Segments, wo die Urgent-Daten enden.

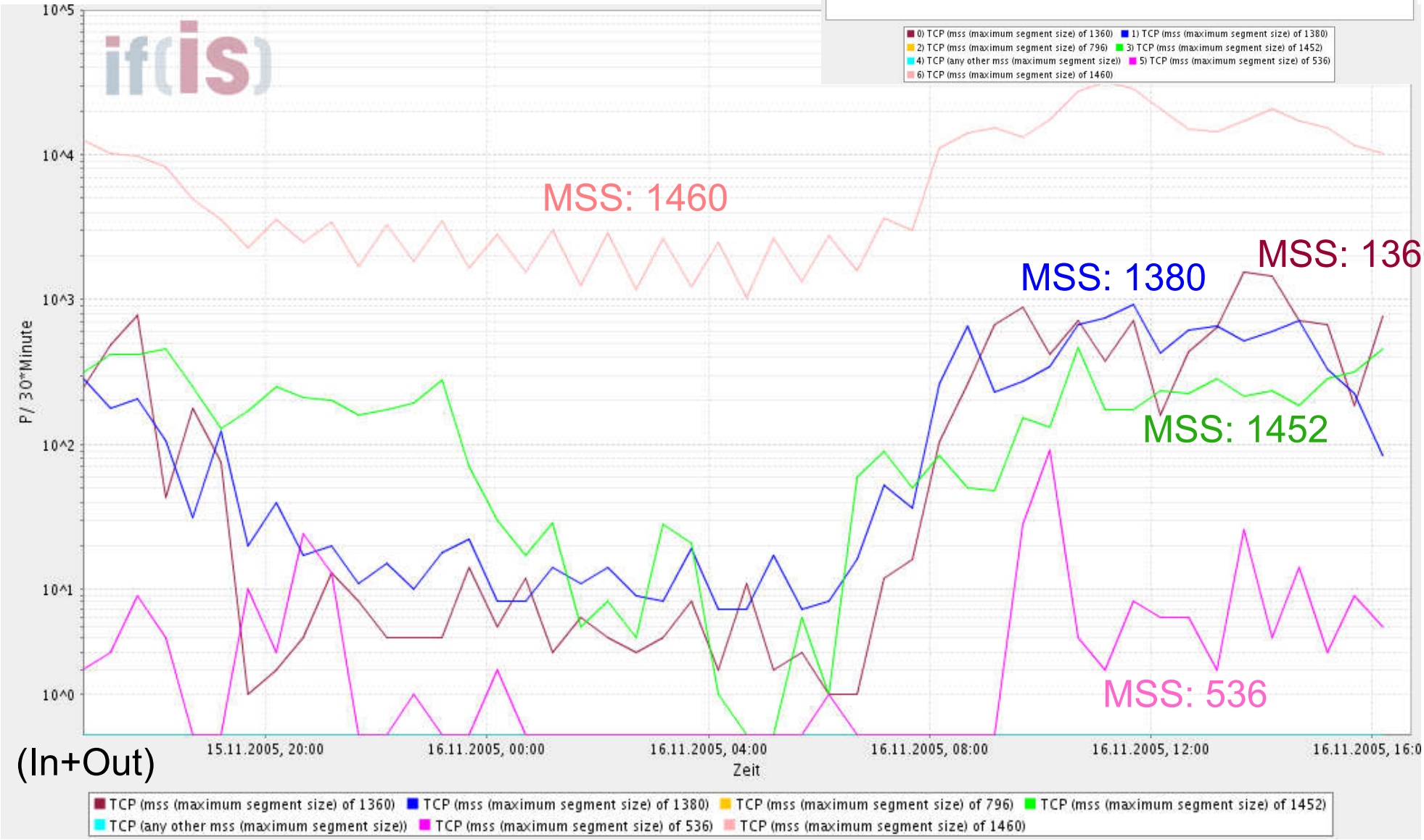
TCP - Transmission Control Protocol
→ Feldelemente des TCP-Headers (5/5)



- **Options**
  - Feldlänge: 24
  - **Beschreibung**
    - Mit dem Option-Feld wird die maximale Größe des TCP-Segments zwischen Sender und Empfänger abgestimmt. Dies ist zur Ausnutzung der maximalen Bandbreite eines Netzes wichtig. Z.B:
      - **MSS (Maximum Segment Size)**
        Legt die maximale Segmentgröße fest, die verarbeitet werden kann.
        Falls ein Rechnersystem diese Option nicht verwendet, werden 536 Byte als die Standardgröße für Nutzdaten verwendet. Die maximale Segmentgröße muss nicht in beiden Richtungen gleich sein.
      - **WSopt (Window Scale)**
        Dieser Skalierungsfaktor wird beim Verbindungsaufbau ausgehandelt.
        Er bestimmt den Faktor mit dem der Wert im Window-Feld multipliziert wird.
        Der Faktor kann maximal den Wert 14 annehmen - Window folglich 1 Gigabyte.
      - **SACK (Selective Acknowledgment)**

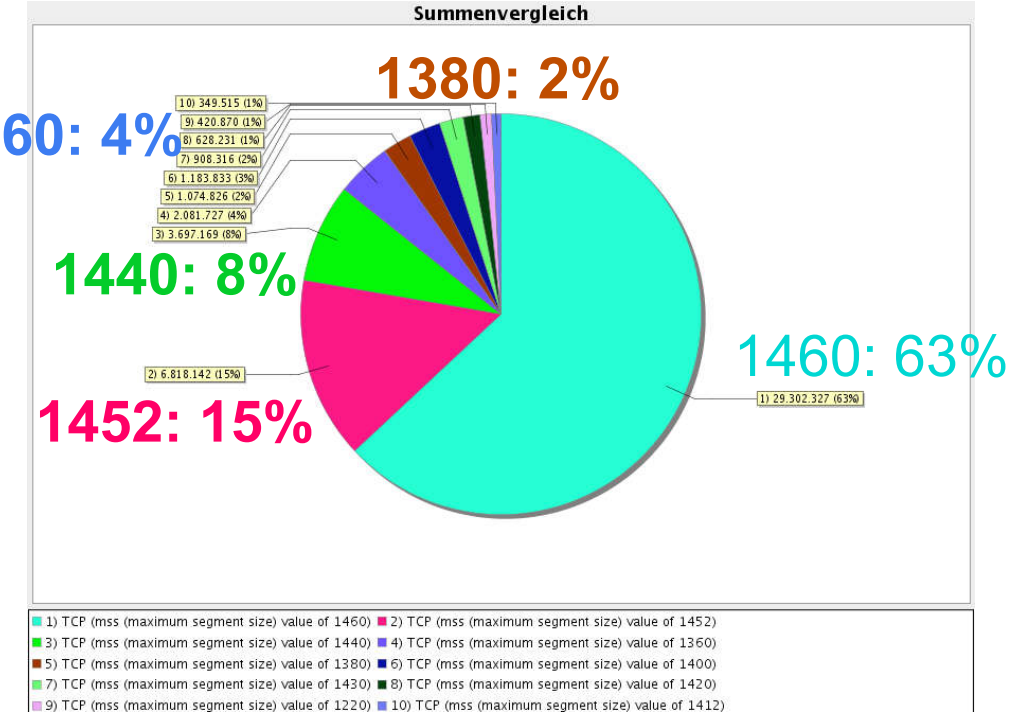
IAS: FB Informatik
→ Option: MSS



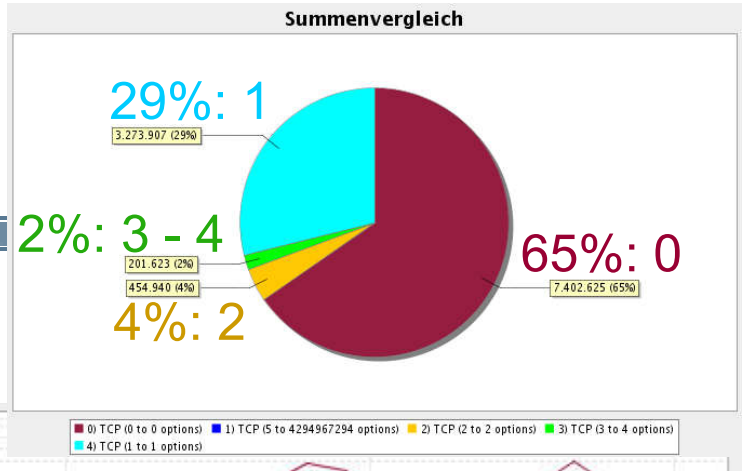


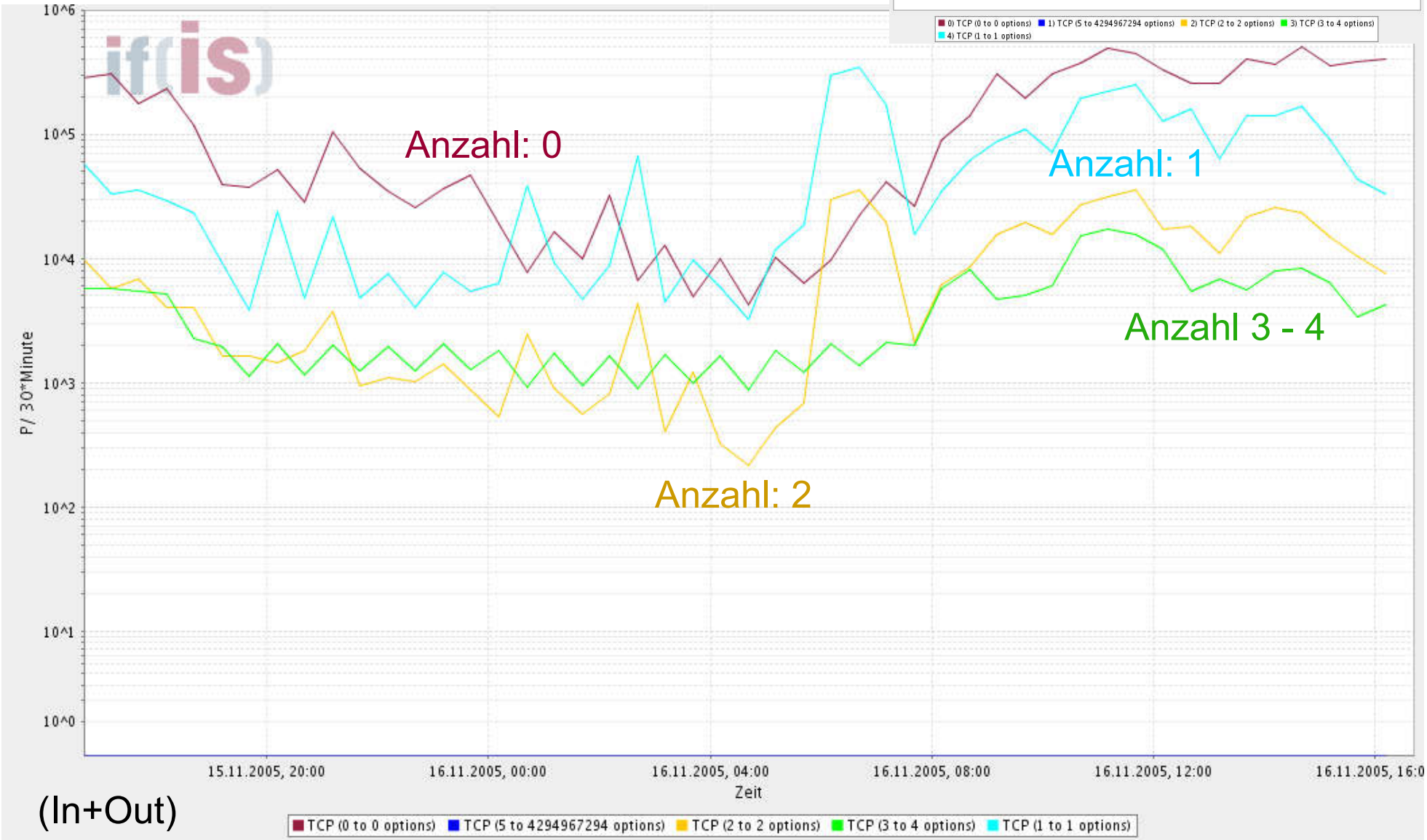
TCP-Header Option: MSS
→ Ergebnisse

- 1460 → MTU 1500: Ethernet, Kabel (UnityMedia), UMTS (O2)
- 1452 → MTU 1492: PPPoE/DSL (Telekom, Alice, Versatel, 1&1)
  → DSL-Nutzer (15 % Summe) →
- 1440 → MTU 1480: ???
- 1360 → MTU 1400: AOL DSL, Compuserve, GRE, IPSEC, PPTP, Modem, UMTS (Base/E-Plus)
- 1380 → MTU 1420: QSC DSL, SAT, WAN, UMTS (Base/E-Plus), ZIM WLAN



IAS: FB Informatik
→ Option (Anzahl)





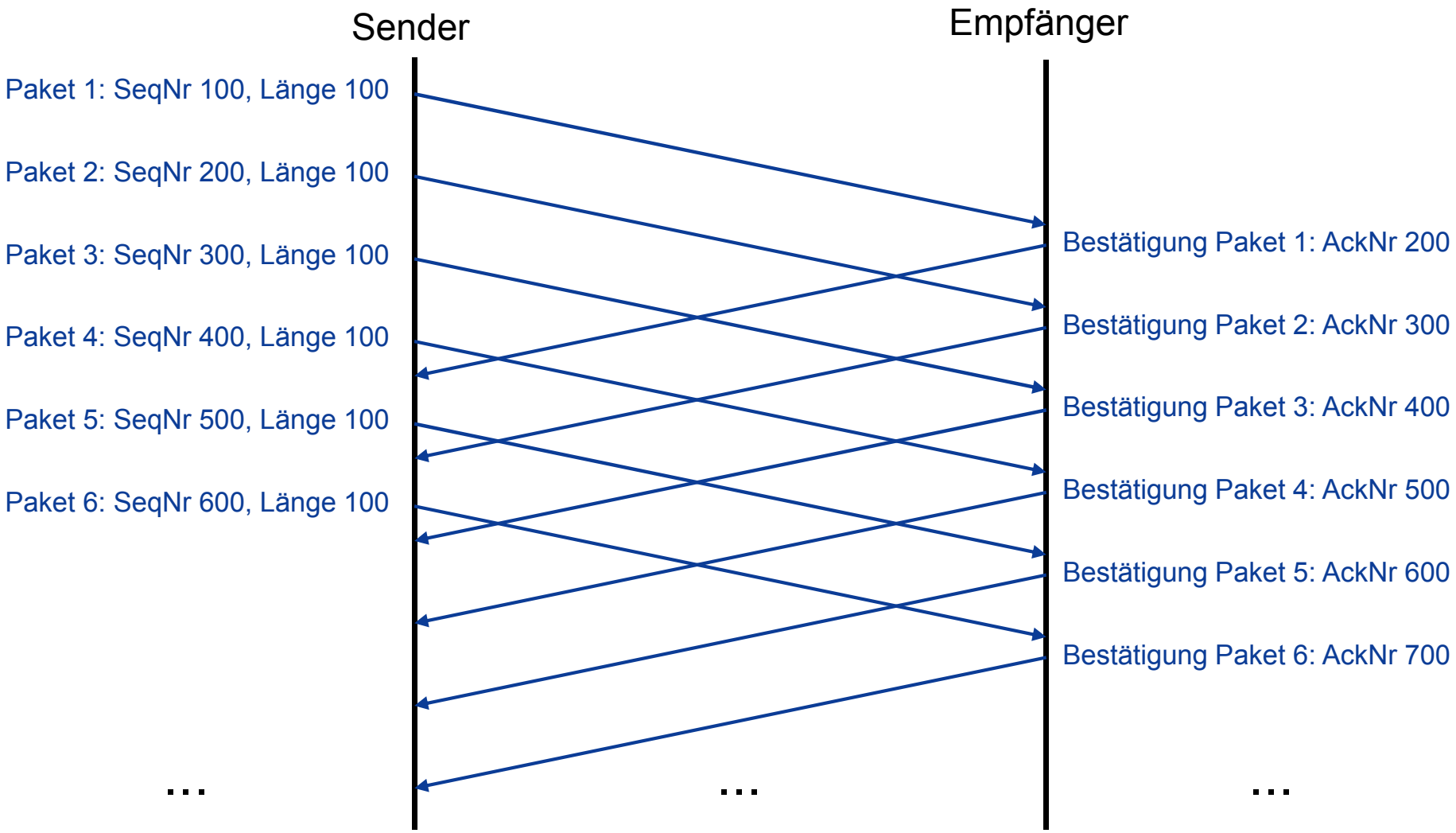
(In+Out)

TCP - Transmission Control Protocol
→ Bestätigung von Daten

- Der Empfänger ordnet empfangene Daten anhand ihrer Sequenznummer, um eine exakte Kopie des gesendeten Datenstroms herzustellen.
- Zu jeder Zeit hat der Empfänger 0 oder mehr Octets (vom Streamanfang zählend) kontinuierlich korrekte Daten empfangen.
- Es können aber auch Teile von Daten korrekt vorhanden sein, die außer der Reihe empfangen wurden.
- Der Empfänger bestätigt aber immer nur das letzte zusammenhängende, korrekt empfangene Paket.
- Dies wird dadurch gekennzeichnet, dass der Empfänger mit der **Acknowledge-Number** (Quittung- bzw. Bestätigungsnummer)die Sequenznummer des nächsten zu empfangenden Octets angibt.

TCP - Transmission Control Protocol
→ Bestätigung von Daten: Beispiel

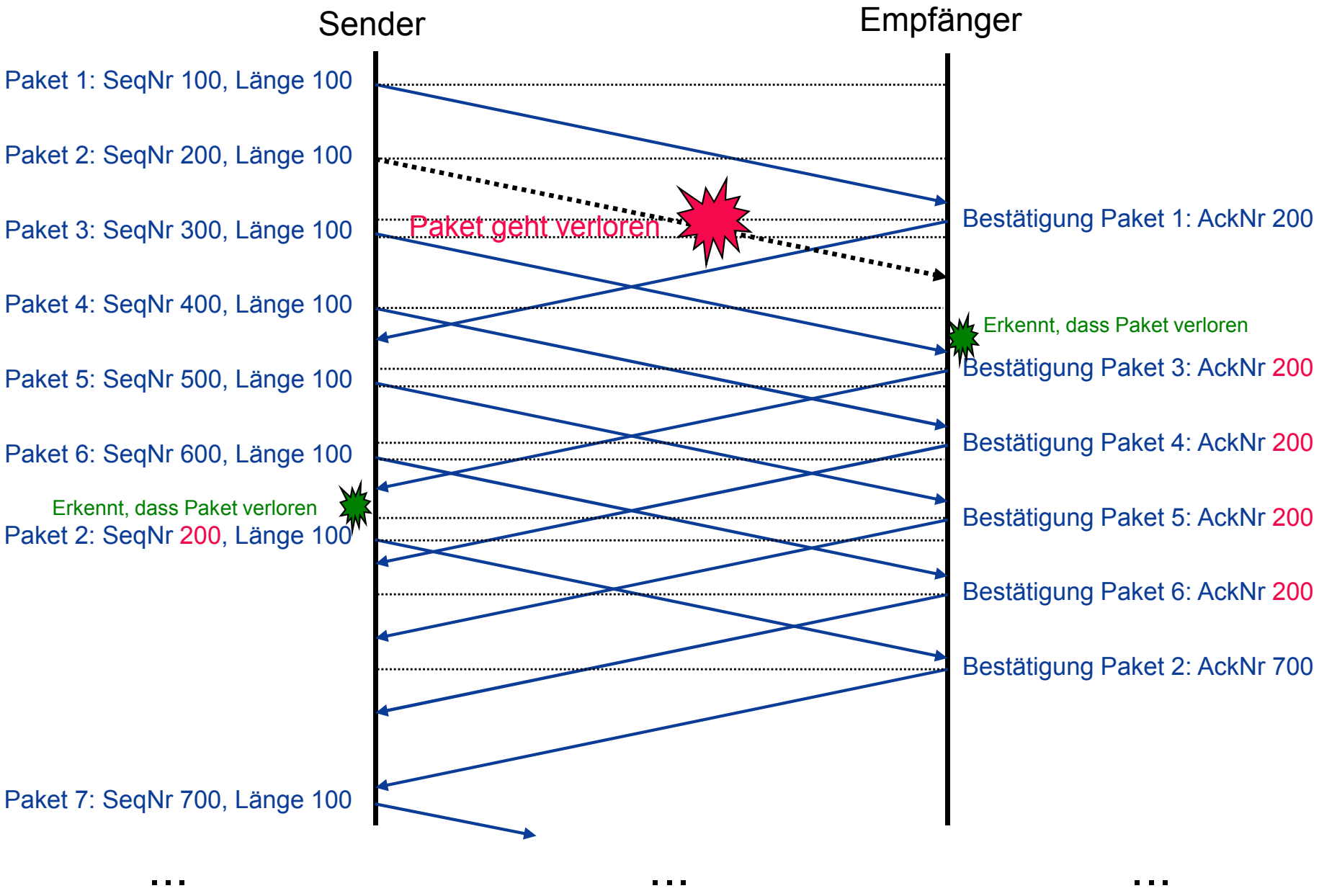
- Ein Sender beginnt bei der Sequenznummer 100 mit einer Window-Size von 500 Octets zu senden.



TCP - Transmission Control Protocol
→ Bestätigung von Daten: Beispiel

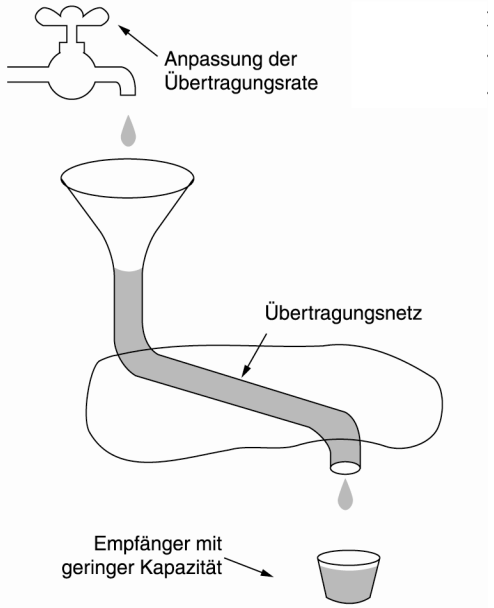
- Geht z.B. das Paket Nr. 2 verloren, so sendet der Empfänger weiterhin Bestätigungen mit der Acknowledge-Nummer 200, da dies die nächste folgende Sequenznummer des Octets ist, die auf den bisher vorliegenden korrekt empfangenen Datenbereich folgt.
- Der Empfänger sendet die AckNr 200 auch, wenn weitere Pakete korrekt empfangen werden.
- Wenn im Sender das Timeout für die Bestätigung des 2. Paketes und der folgenden erfolgt, muss dieser entscheiden, ob er alle 5 Pakete oder nur das erste Paket des aktuellen Fensters verschickt.
- Alle 5 zu wiederholen ist möglich, wäre natürlich nicht sehr effektiv.
- Wiederholt der Sender nur Paket Nr. 2 und hat der Empfänger die Pakete Nr. 3-6 korrekt empfangen, antwortet der Empfänger mit der AckNr 700 und das Sendefenster kann im Sender um 500 Octets verschoben werden.

TCP - Transmission Control Protocol
→ Bestätigung von Daten: Beispiel



TCP - Transmission Control Protocol
→ Flusskontrolle bei TCP - Zusammenfassung

- Bei der Kommunikation entsteht das Problem, dass die Menge der übertragenen Daten an die **Aufnahmefähigkeit des Empfängers** angepasst werden muss.
- Die übertragene Datenmenge sollte nicht größer sein als die Datenmenge, die der Empfänger aufnehmen kann.
- Die Menge der übertragenen Daten muss zwischen den kommunizierenden Rechnern entsprechend abgestimmt werden.
- Diese Abstimmung wird als **Flusskontrolle (Flow Control)** bezeichnet.
- Für die Flusskontrolle nach der Sliding-Window Technik werden folgende Angaben im TCP-Header verwendet:
  - Sequenzummer
  - Acknowledge-Number (Quittung- bzw. Bestätigungsnummer)
  - Window-Größe

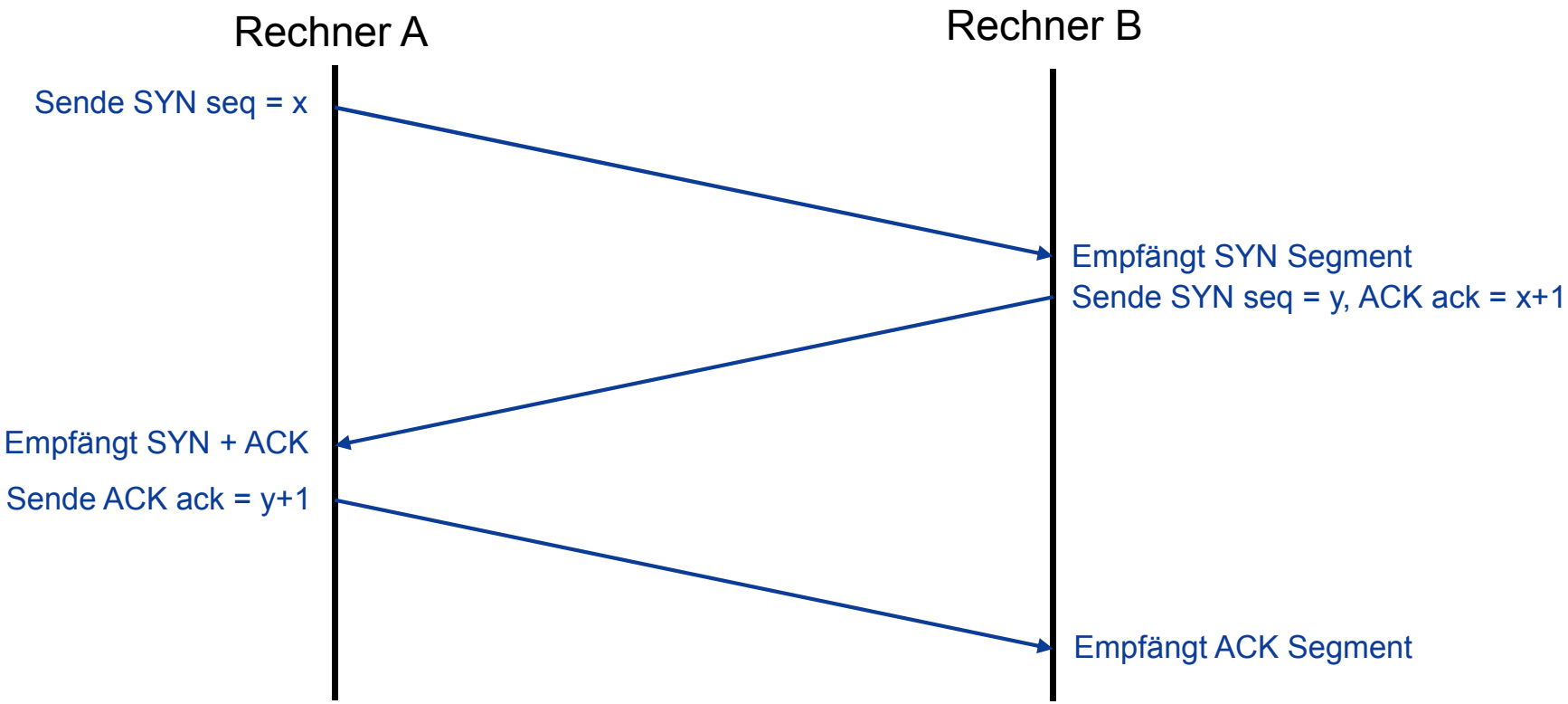


0	4	10	16	24	31
Source Port			Destination Port		
Sequence Number					
Acknowledgement Number					
HLEN	Reverved	Code Bits	Window		
Checksum			Urgent Pointer		
Options (optional)				Padding	

TCP - Transmission Control Protocol
→ Ports und Verbindungen (Sockets)

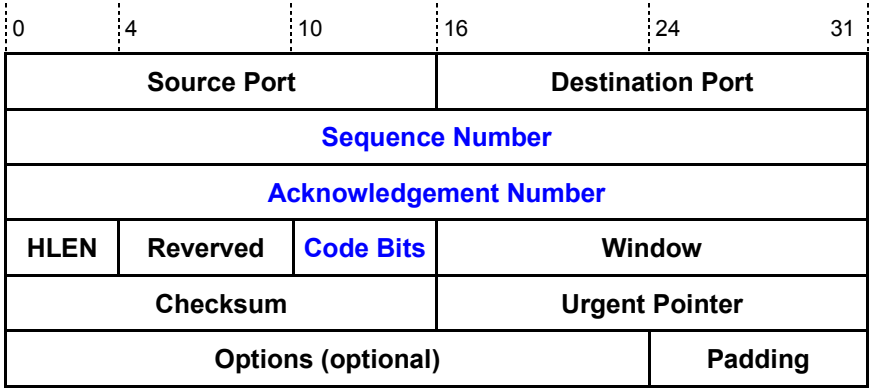
- Wie UDP benutzt TCP den Mechanismus der Ports, um einzelne Prozesse voneinander zu unterscheiden.
- Die Ports bei TCP sind allerdings um einiges komplexer, da es im Gegensatz zu UDP virtuelle Verbindungen gibt.
- Eine Verbindung ist dabei definiert durch ein Paar von Endpunkten.
- Ein **Endpunkt einer Verbindung** setzt sich aus der **IP-Adresse** und der **Portnummer** zusammen, z.B. (128.10.2.30, 2000).
- Eine Verbindung setzt sich aus zwei solcher Endpunkte zusammen, z.B. durch (128.10.2.30, 2000) und (53.1.182.99, 1500).
- Zu einem Endpunkt (oder auch Socket) können zu einer Zeit mehrere Verbindungen existieren.
- Dies ist möglich, da TCP eingehende Nachrichten mit der Verbindung statt nur mit der Portnummer in Zusammenhang stellt.
- Ein Programmierer benötigt für ein Programm, welches mit mehreren anderen Applikationen kommunizieren will, lediglich eine Portnummer und nicht für jede Verbindung eine andere.

TCP - Transmission Control Protocol
→ Aufbau einer TCP-Verbindung



seq: Sequenz-Nummer
ack: Acknowledge-Nummer

SYN: Synchronisations-Bit
ACK: Acknowledge-Bit



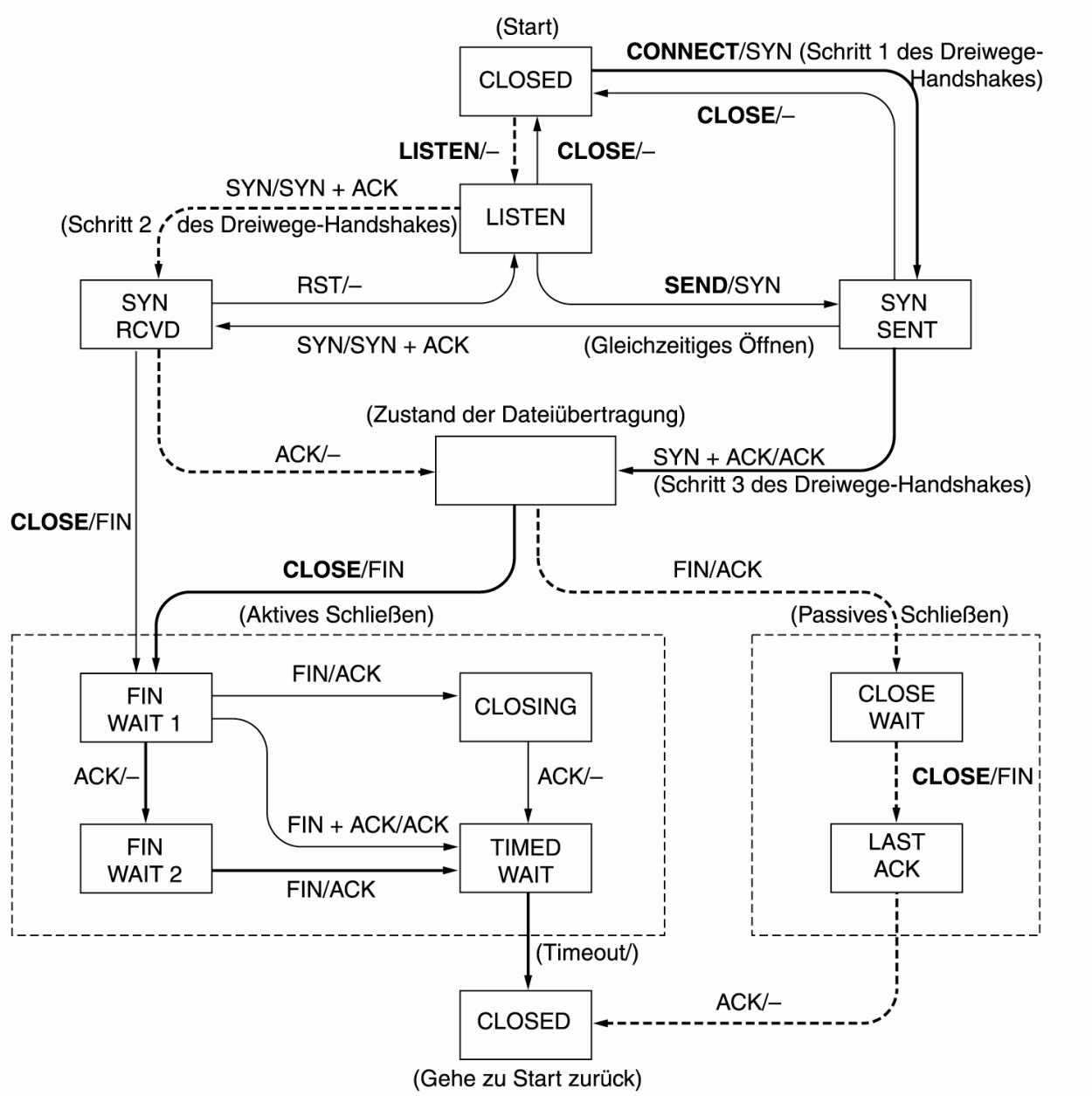
TCP - Transmission Control Protocol
→ Aufbau einer TCP-Verbindung

- Beim Verbindungsaufbau werden zwei, für den Datenaustausch wichtige Voraussetzungen erfüllt:
  - Es wird garantiert, dass **beide Seiten zum Datenaustausch bereit** sind und
  - **beide Seiten haben die Startwerte** der Sequenznummern ausgetauscht.
- Wichtig ist der Austausch der Startwerte vor allem dann, wenn schon das erste Datenpaket, welches geschickt wurde, verloren geht.
- Auf alle folgenden Datenpakete muss der Empfänger mit einer Bestätigung und der entsprechenden ack (Nr.) antworten.
- Dies ist immer die ack (Nr.), die der seq (Nr.) des ersten Octets des ersten, verlorengegangenen Pakets entspricht.
- Für den Fall nach der Verbindungsaufnahme ist die ack (Nr.) bei Verlust des ersten Pakets immer x+1 (falls A Sender und B Empfänger ist).

TCP - Transmission Control Protocol

→ Auf- und Abbau einer TCP-Verbindung/Zustandsautomat

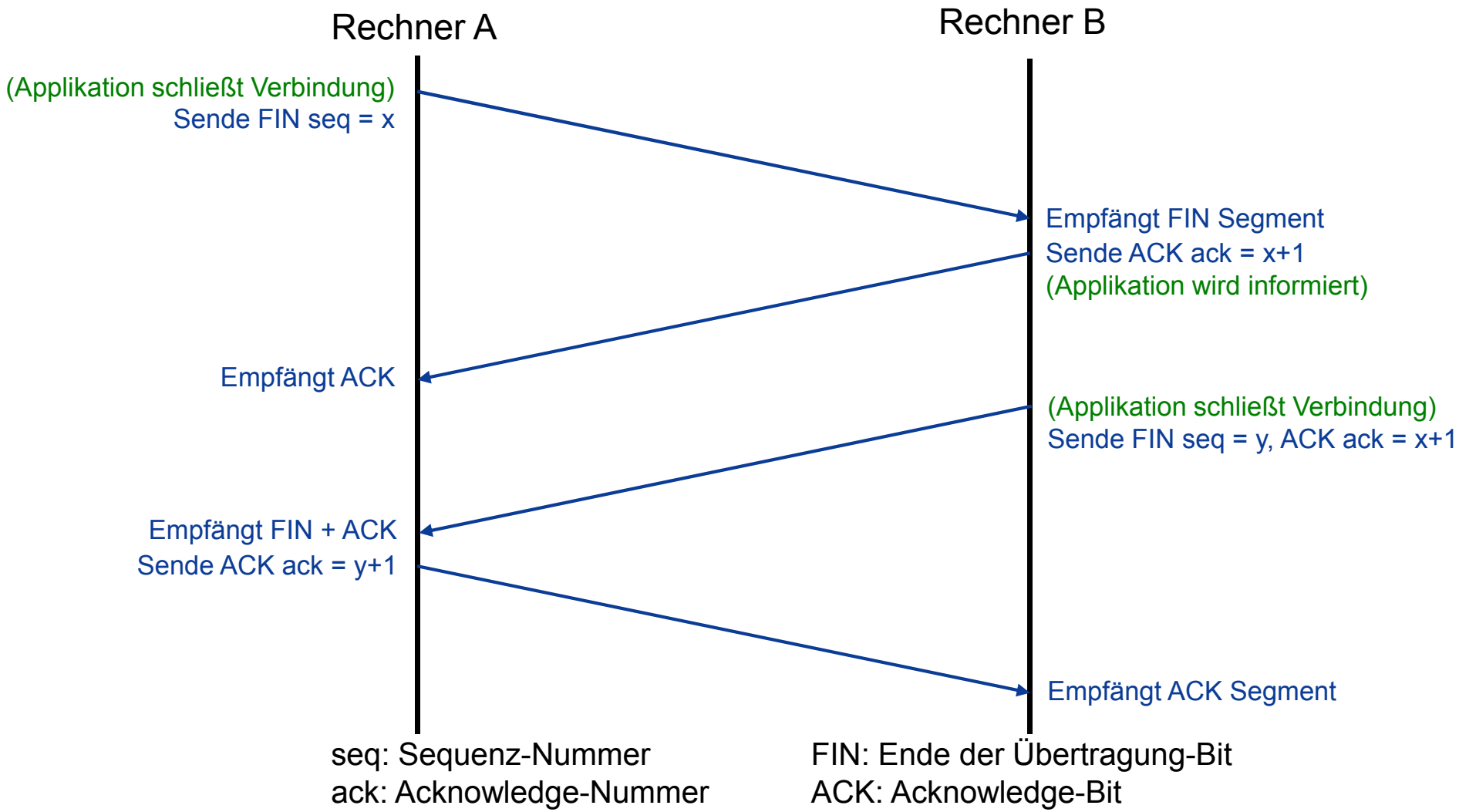
Zustand	Beschreibung
CLOSED	Keine Verbindung aktiv oder anstehend
LISTEN	Der Server wartet auf eine ankommende Verbindung.
SYN RCVD	Eine Verbindungsanfrage ist angekommen. Warten auf Bestätigung
SYN SENT	Die Anwendung hat begonnen, eine Verbindung zu öffnen.
ESTABLISHED	Zustand der normalen Datenübertragung
FIN WAIT 1	Die Anwendung möchte die Übertragung beenden.
FIN WAIT 2	Die andere Seite ist einverstanden, die Verbindung abzubauen.
TIMED WAIT	Warten, bis keine Pakete mehr kommen
CLOSING	Beide Seiten haben gleichzeitig versucht, zu beenden.
CLOSE WAIT	Die Gegenseite hat die Freigabe eingeleitet.
LAST ACK	Warten, bis keine Pakete mehr kommen



TCP - Transmission Control Protocol
→ Abbau einer TCP-Verbindung

- Eine TCP-Verbindung ist eine Full-Duplex Verbindung mit zwei unabhängigen, in gegensätzlichen Richtungen fließenden Datenströmen.
- Diese Richtungen können unabhängig voneinander geschlossen werden.
- Wenn eine Applikation der TCP-Software meldet, dass keine Daten mehr zum Senden vorhanden sind, so wird die Senderichtung beendet.
- TCP beendet dabei das Senden der Daten, wartet bis das letzte Segment bestätigt wurde und leitet dann die Verbindungsabbau-Prozedur ein (siehe Zustandsautomat).
- Ist eine Verbindungsrichtung einmal beendet, so verweigert die TCP-Software die Annahme weiterer Daten, die darüber transportiert werden sollen.
- Die Verbindung in die Gegenrichtung wird auch mit der Prozedur zum Verbindungsabbau beendet.
- Erst wenn beide Verbindungsrichtungen beendet sind, verwirft die TCP-Software die Information der Verbindung und beendet die Kommunikation mit der Applikation.

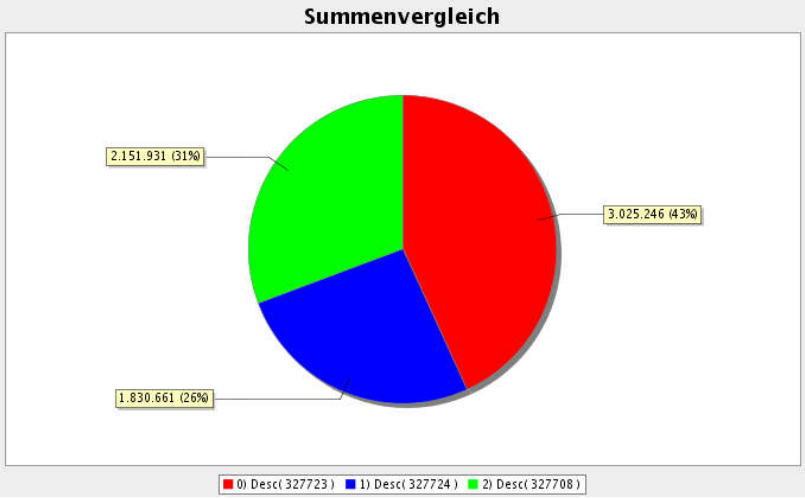
TCP - Transmission Control Protocol
→ Abbau einer TCP-Verbindung



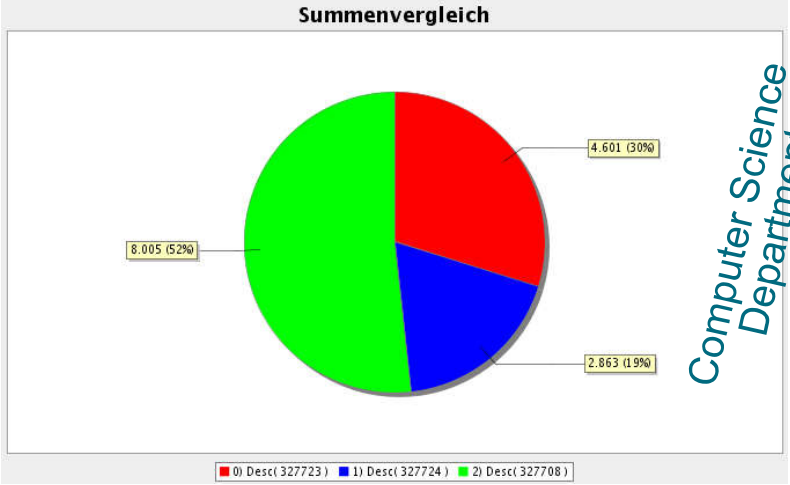
Hinweis: Das FIN-Bit kann auch im letzten Datenpaket gesetzt sein (siehe PM FTP-Datenkanal)

Beispiele von Ergebnissen des IAS
→ Wissensbasis: Zusammenhänge

- **SYN-Scan (Potentielle Angriffsituation)**
  - Vergleich unterschiedlicher Zeiträume
    - Normal: SYN > SYN/ACK > 2xFIN/ACK
  - Diskrepanz: Normalverteilung zu Verteilung in einer Angriffssituation
    → Angriffssituationserkennung



SYN
(31% - 52%)
SYN/ACK
(26% - 19%)
FIN/ACK
(43% - 30%)

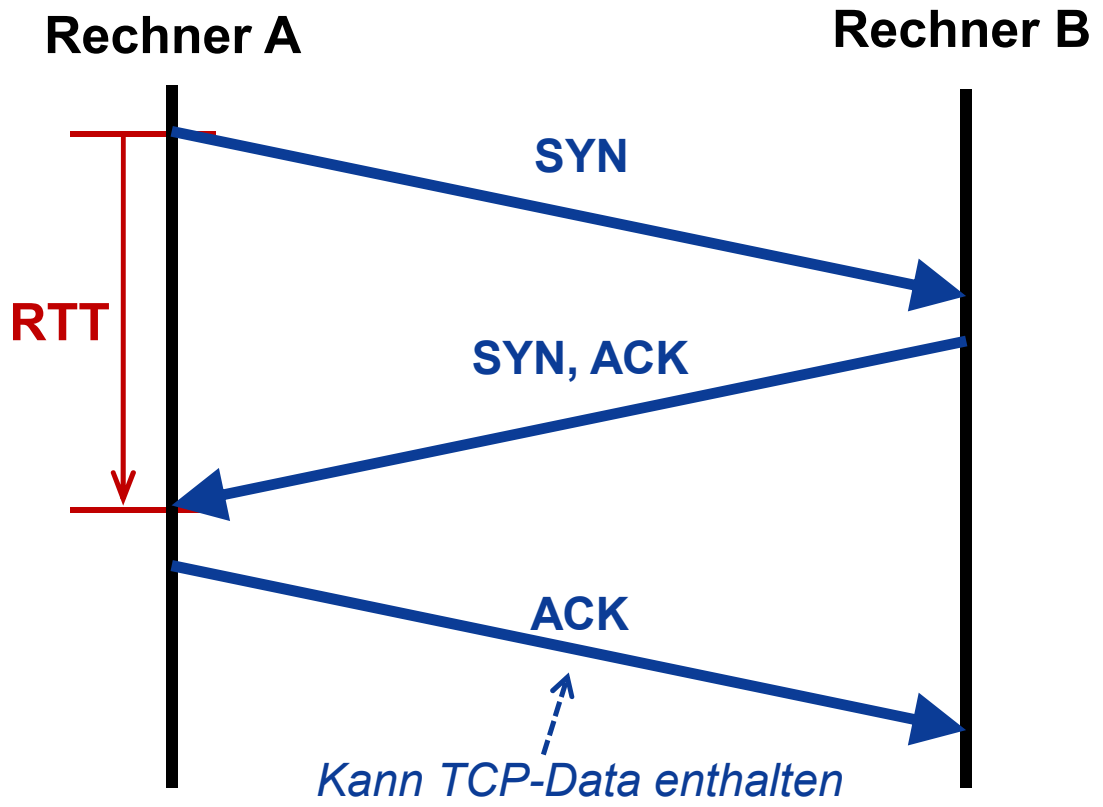


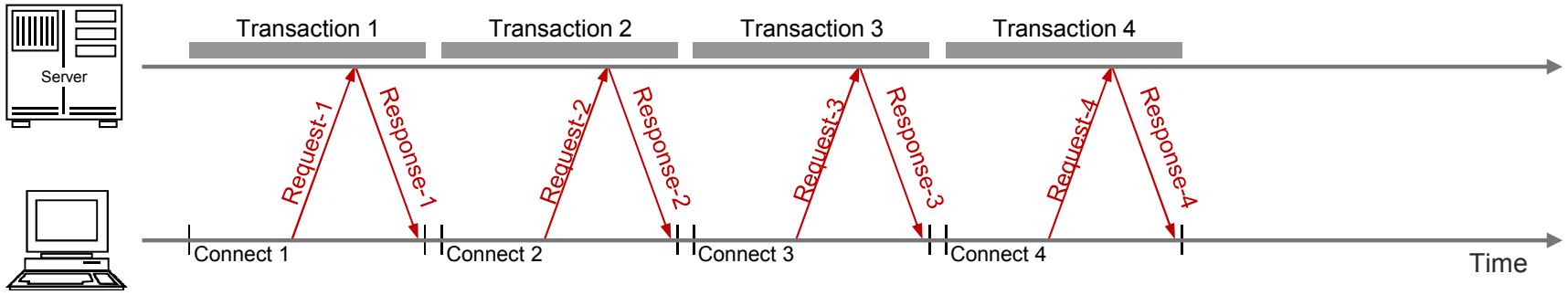
Normale Verteilung

Annormale Verteilung

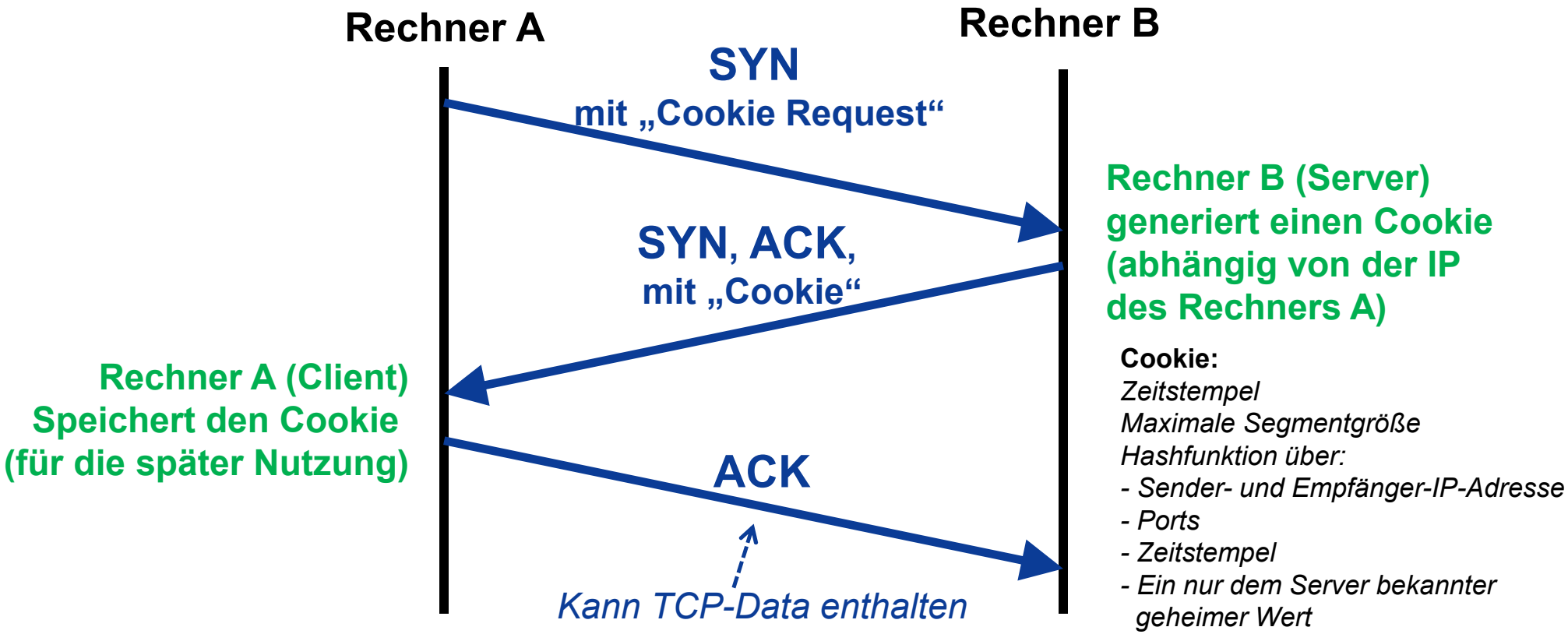
TCP Fast Open (TFO)
→ Motivation

- „Hohe Kosten“ für den Verbindungsaufbau bei TCP
- Viele kleine TCP-Session (HTTP, …)



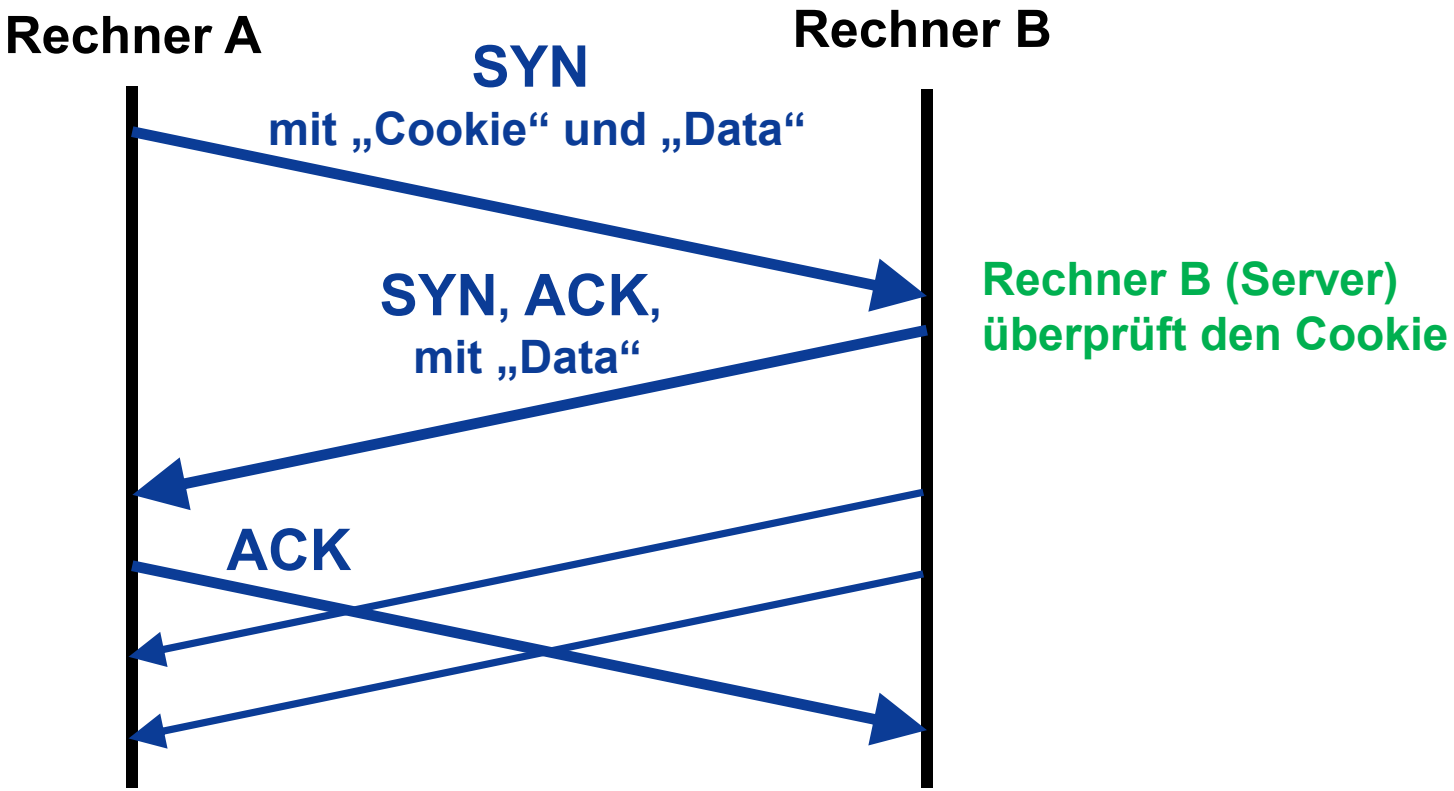


TCP Fast Open (TFO)
→ Session Startup



- (Security-)Cookie, um den Kommunikationspartner zu authentisieren
- Cookie verhindert SYN-Flood von spoofed Paketen
- Cookie der TCP-Instanz

TCP Fast Open (TFO)
→ Fast Lane



- 10 % bis 40 % Einsparung (einen RTT)
- Ist bei Linux ab Kernel 3.13 standardmäßig aktiviert

TCP Fast Open (TFO)
→ Nutzung

- SYN (TFO) = 10,4 %
- SYN/ACK (TFO) = 6 %

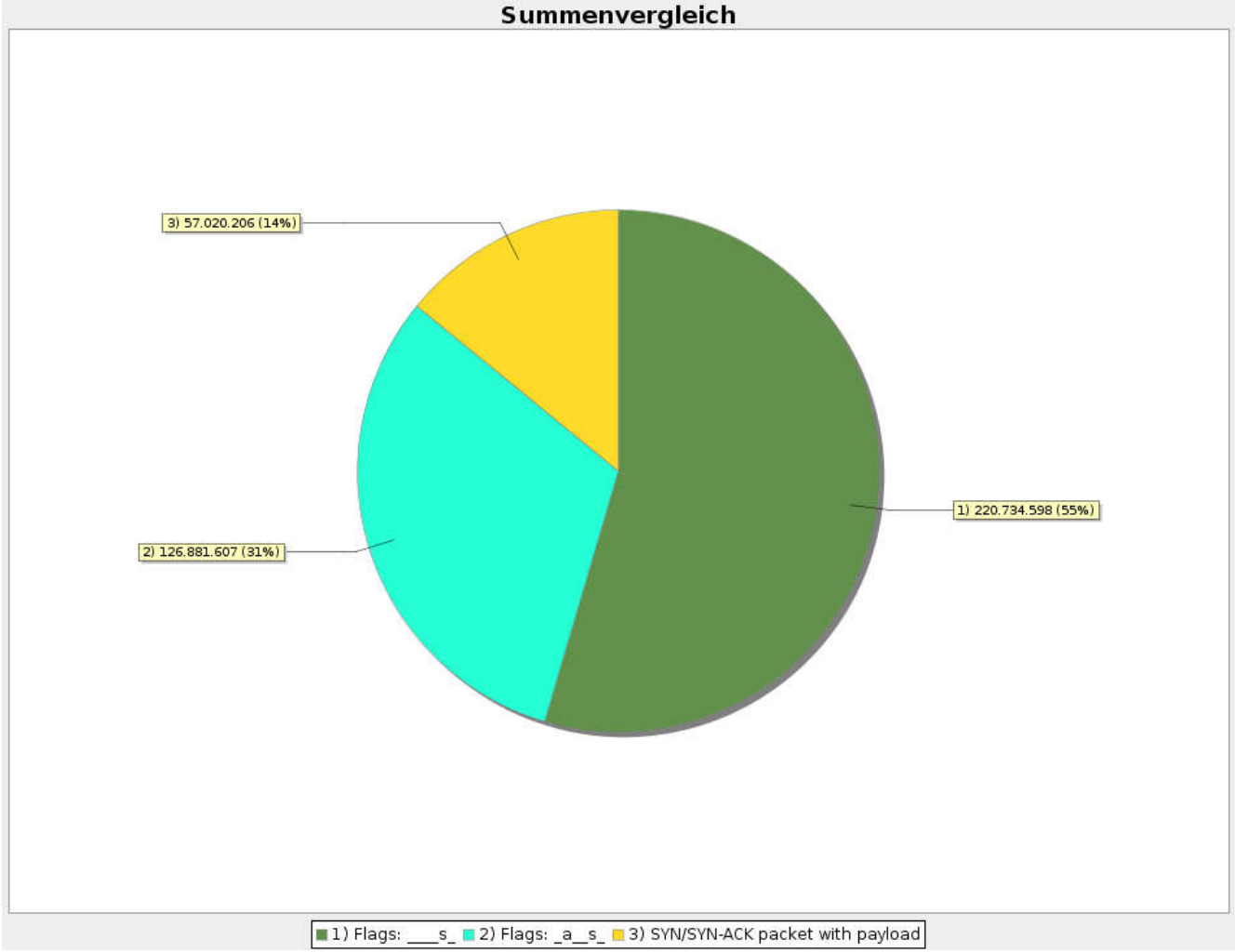
Hinweis:

1) und 2) = 100 %

63,5 % SYN

36,5 % SYN/ACK

16,6 % TFO



TCP - Transmission Control Protocol
→ Portnumern

- Wie auch bei UDP gibt es bei TCP fest vereinbarte Portnummern (reservierte Portnummern oder „well known ports“).
- Bei TCP sind die ersten 256 (0 bis 255) Portnummern reserviert.
- Beispiele:

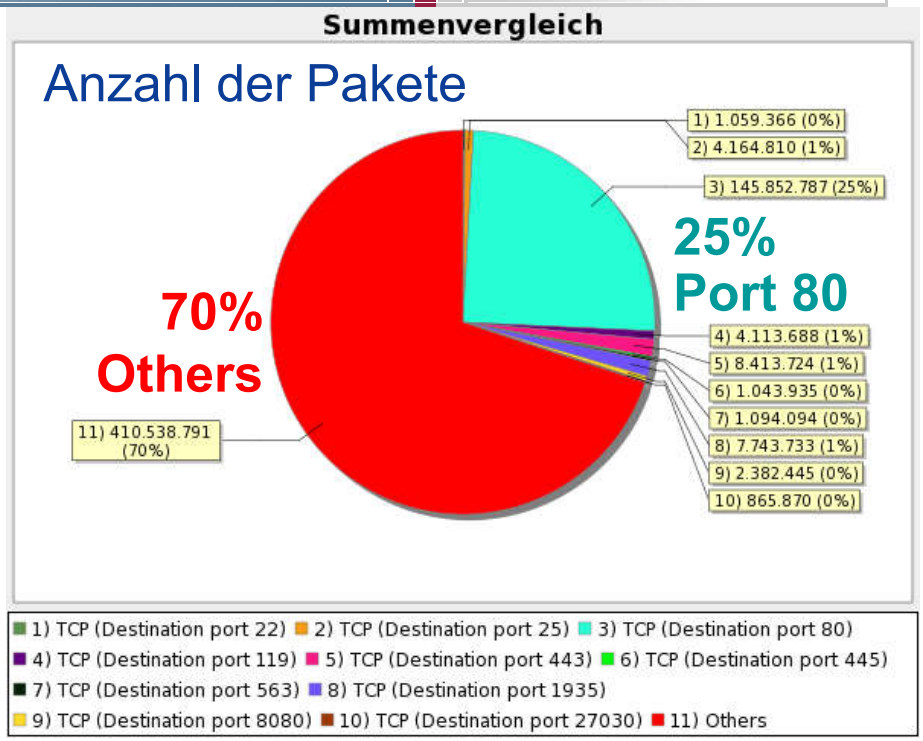
Portnummer	Bezeichnung	Beschreibung
20	FTP	File Transfer Protocol - Daten
21	FTP	File Transfer Protocol - Kommandos
23	Telnet	Terminal Connection
25	SMTP	Simple Mail Transport Protocol
53	DNS	Domain Name Service
80	HTTP	Hypertext Transfer Protocol
103	X.400	X.400 Mail Service over IP
110	POP3	Post Office Protocol
113	Auth	Authentication Service
143	IMAP	Internet Message Access Protocol

https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

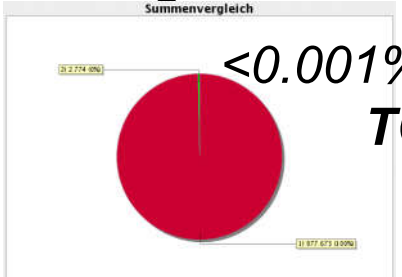
- 256 - 1023 reserviert für priviligierte Benutzer (z.B. Superuser in UNIX)
- 1024 - 5000 transienter Bereich; Portnummern werden dynamisch generiert
- 5001 - 65553 Portnummern frei für Anwendungen

TCP Top10 Destination Ports (alle)
(TCP ca. 75 % der Σ des IPv4-Verkehrs)

1. 25% HTTP (80)
2. 1% HTTPS (443)
3. 1% RTMP (1935)
   (Flash Real Time Messaging Protocol)
4. 1% NNTP (119)
   (→ Usenet → Downloads)
5. 1% SMTP (25)
6. <1% HTTP (8080)
7. <1% NNTPS (563)
   (→ Usenet → Downloads)
8. <1% SSH (22)
9. <1% SMB (445)
   (bedenklich → Exploitversuche, direkt angeschlossene MS-Systeme?)
10. <1% Steam (27030) (→ Gaming)



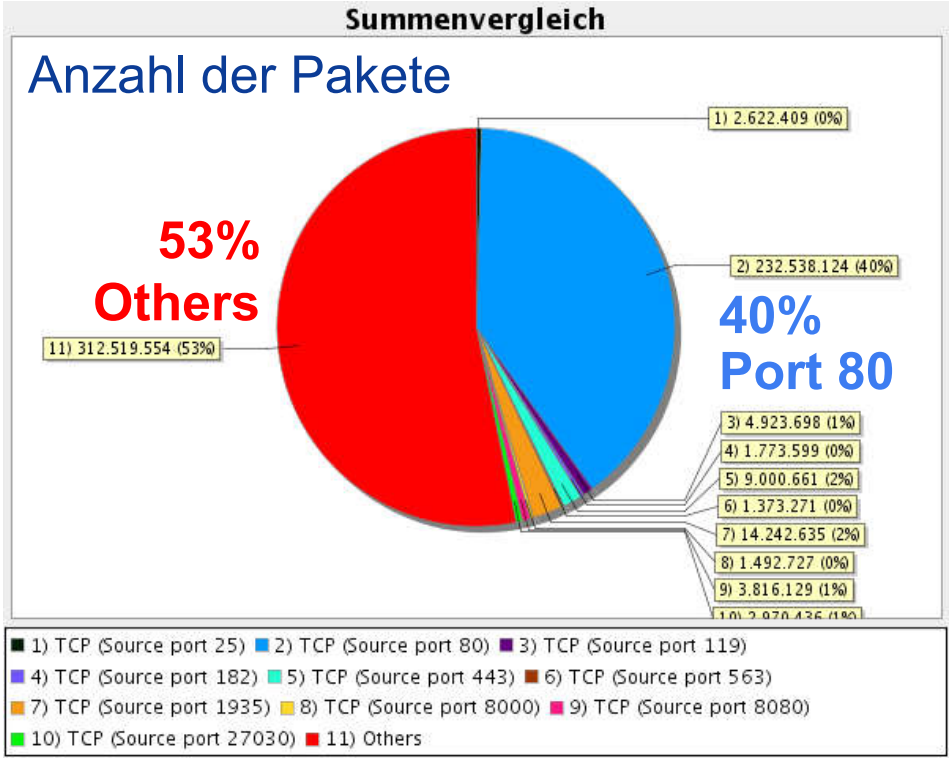
(Top10 = 30%)



<0.001% 445 SRC (SYN/ACK)
TCP-Verbindung (2.2774)

TCP Top10 Source Ports (alle)
(TCP ca. 75 % der Σ des IPv4-Verkehrs)

		DST/SRC
1.	40% HTTP (80)	*1 zu 1,6*
2.	2% RTMP (1935) (Flash Real Time Messaging Protocol)	*1 zu 1,8*
3.	2% HTTPS (443)	*1 zu 1,1*
4.	1% NNTP (119) (→ Usenet → Downloads)	*1 zu 1,2*
5.	1% HTTP (8080)	*1 zu 1,6*
6.	1% Steam (27030) (→ Gaming)	*1 zu 3,4*
7.	<1% SMTP (25)	*1,6 zu 1*
8.	<1% Unisys Audit SITP (182)	
9.	<1% HTTP (8000)	
10.	<1% NNTPS (563) (→ Usenet → Downloads)	*1 zu 1,3*

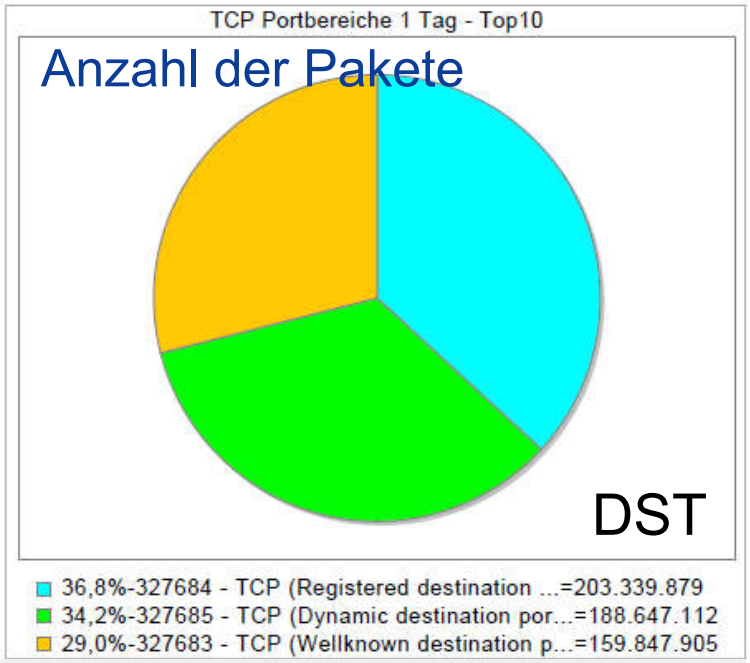


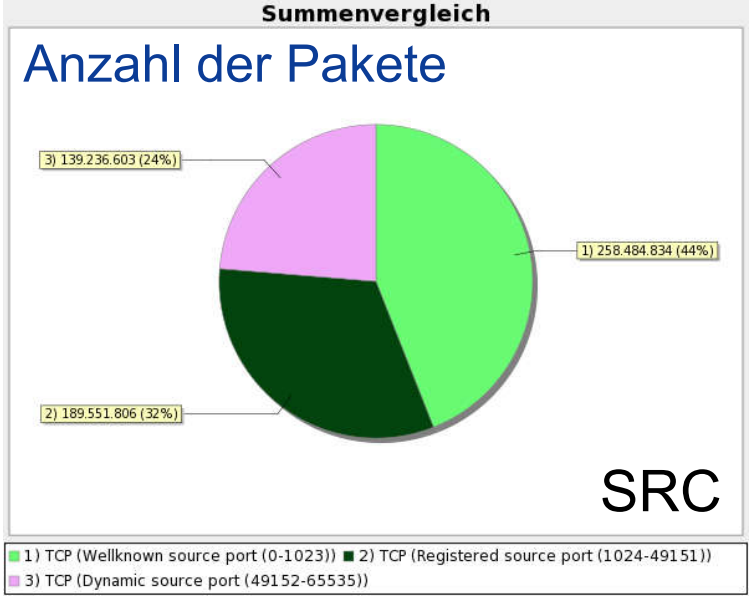
(Top10 = 47%)

TCP DST/SRC Ports
(TCP ca. 75 % der Σ des IPv4-Verkehrs)

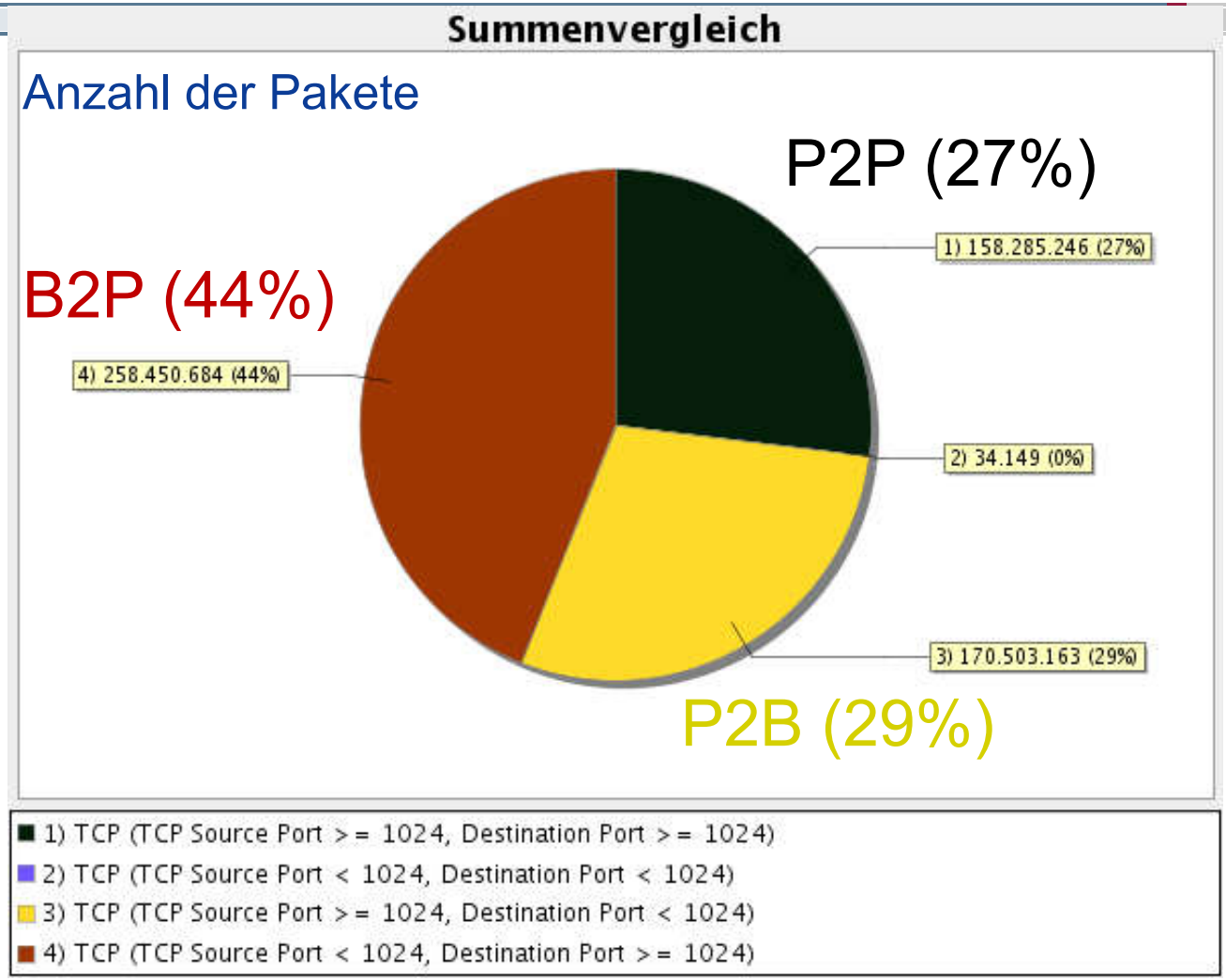
Well known Ports (**WP**)	0 - 1023
Registered Ports (**RP**)	1024 - 49151
Dynamic Ports (**DP**)	49152 - 65535

- **DST Ports**
  - 29% Well known **(P2B)**
  - 36,8% Registered
  - 34,2% Dynamic
- **SRC Ports**
  - 44% Well known **(B2P)**
  - 32% Registered
  - 24% Dynamic





P2P Counter
→ Heuristik



- Src >= 1024 and Dst >= 1024 (« P2P ») - client-to-client
- Src < 1024 and Dst < 1024 **(« B2B »)** - server-to-server
- Src >= 1024 and Dst < 1024 **(« P2B »)**
- Src < 1024 and Dst >= 1024 **(« B2P »)**

TCP Betriebssystem SRC Ports
→ Betriebssystem-Strategie

- Windows XP and Previous: 1024 - 4999
- Windows Vista and Server 2008: 49152 - 65352
- Apple Mac OS: 49152 - 65535
- Linux (2.4 kernel), Android: 32768 - 61000
- Free BSD: 1024 - 65535 (assigned randomly)
- Solaris: 32768 - 65535

- **Zahlen für TCP Dynamic SRC Ports:**
  - 1024 - 4999 (Windows alt): 114.302.605 **(44,25 %)**
  - 32768 - 49151 (Linux, Android): 33.864.740 **(13,10 %)**
  - 49152 - 61000 (Windows neu + Apple): 86.943.628 **(33,66 %)**
  - 61001 - 65352 (Verschiedene): 23.179.307 **(8,97 %)**

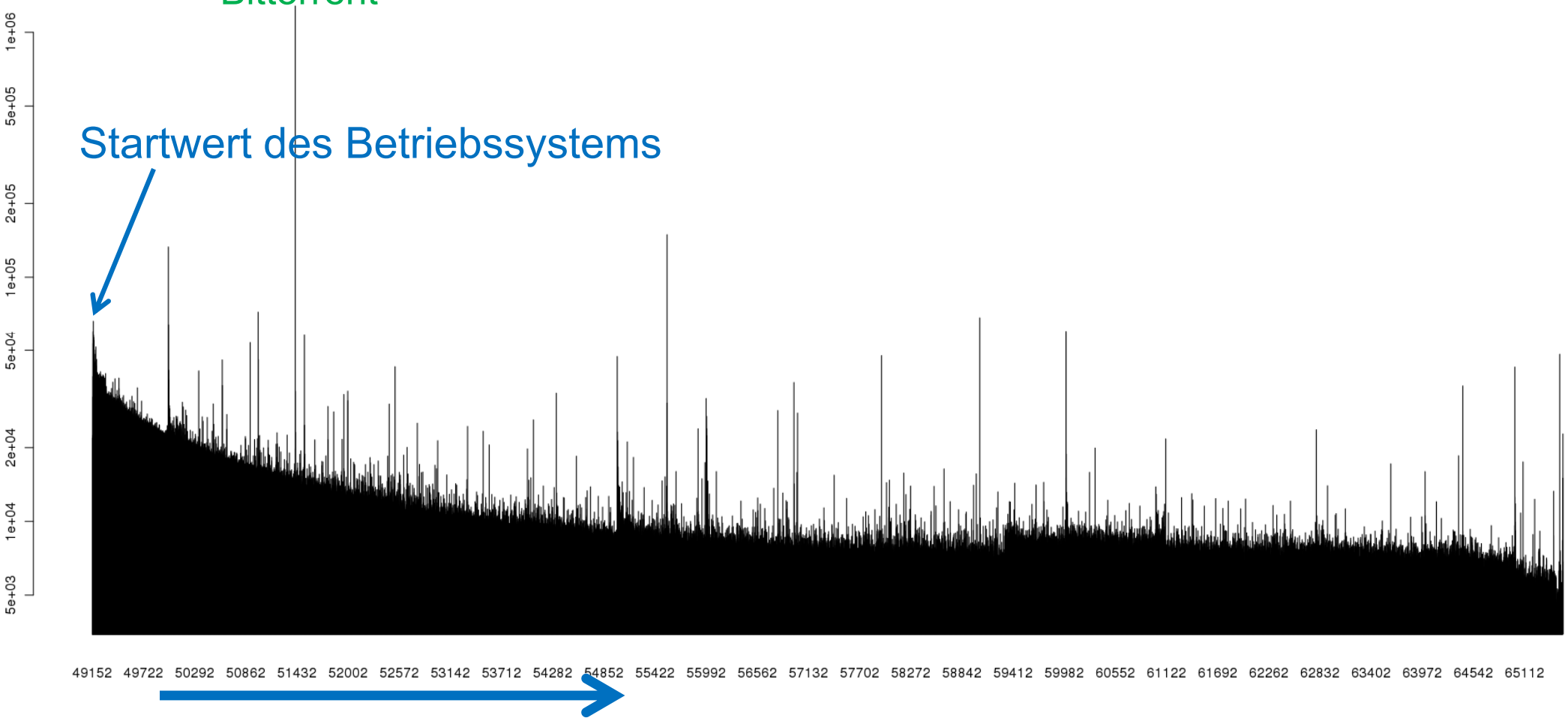
- **Zahlen für UDP Dynamic SRC Ports:**
  - 1024 - 4999 (Windows alt): 14.294.882 **(13,50%)**
  - 32768 - 49151 (Linux, Android): 41.700.713 **(39,38%)**
  - 49152 - 61000 (Windows neu + Apple): 37.576.523 **(35,48%)**
  - 61001 - 65352 (Verschiedene): 12.323.963 **(11,64%)**

TCP Dynamic SRC Ports (49152-65353)

→ Betriebssystem-Strategie (Linux, Windows neu, ...)

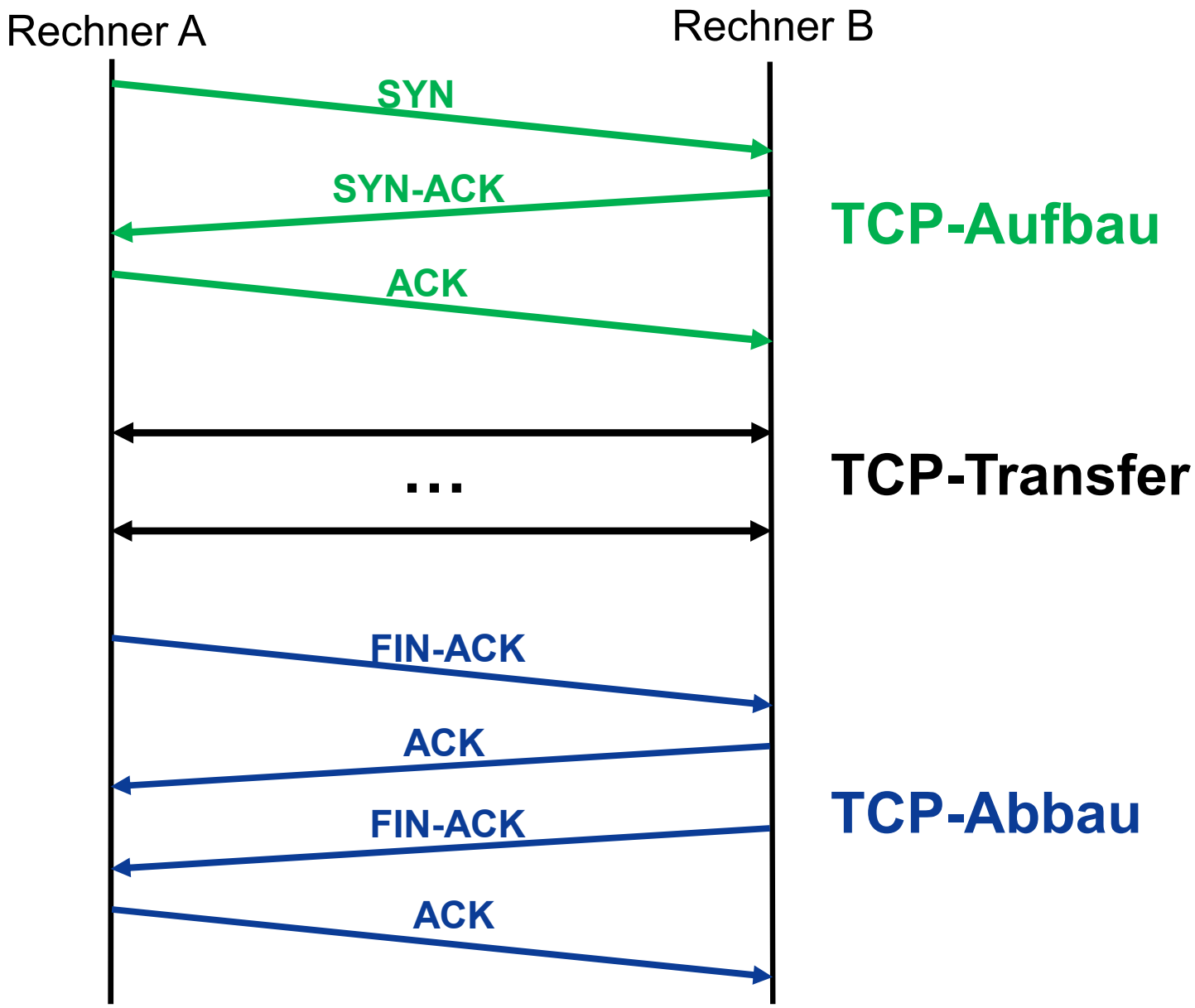
IANA Ephemeral Port Range (33%)

Port 51413
Bittorrent



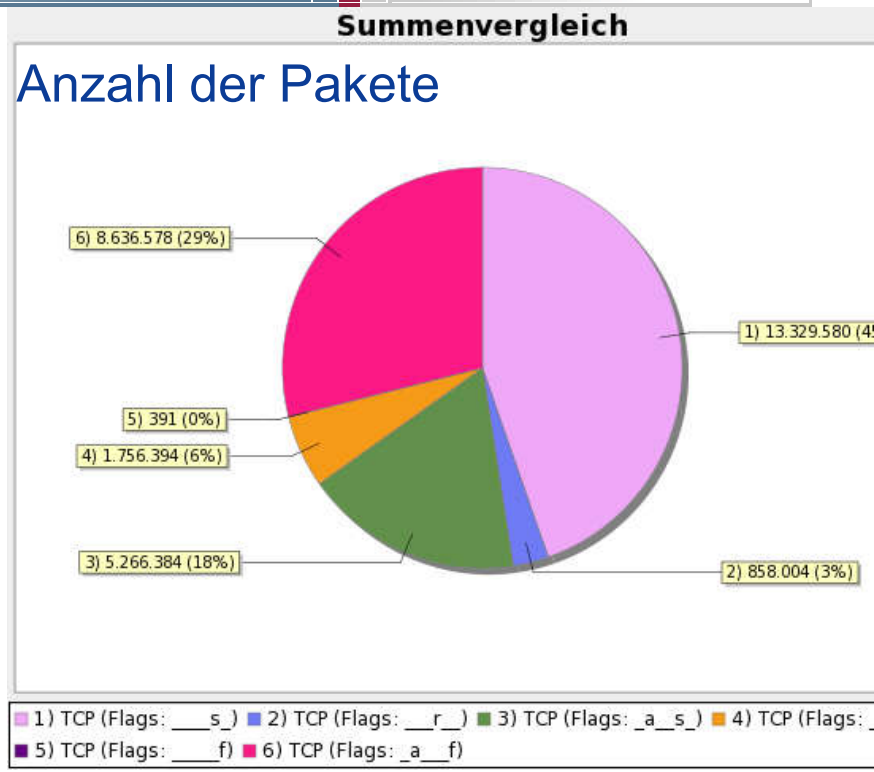
Je mehr Verbindungen das Betriebssystem aufbaut,
desto höher wird der Ephemeral Port

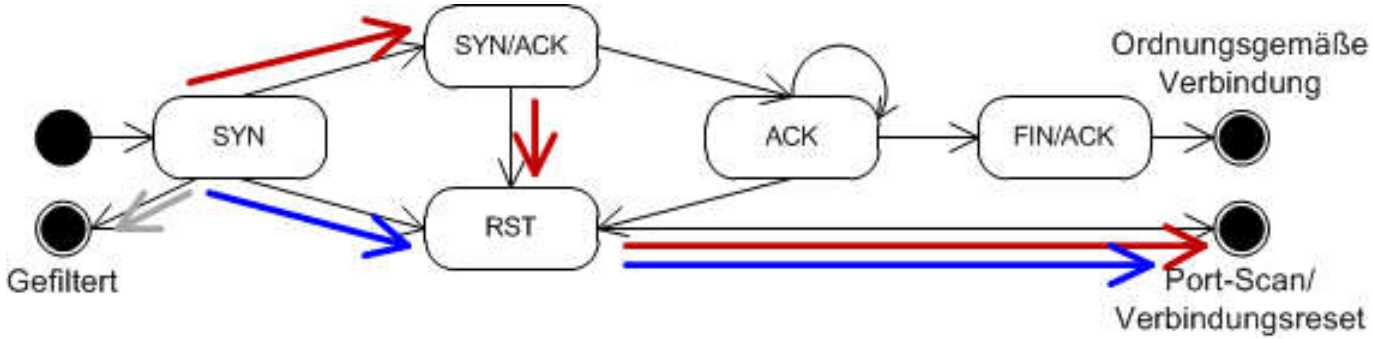
Analyse TCP
→ TCP-Kommunikation: Übersicht

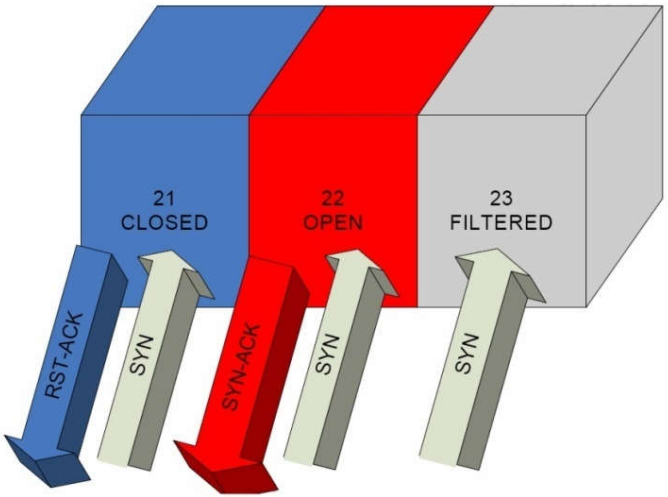


TCP-Header „Code Bits“-Feld
→ Verbindungsauf- und abbau

- 45% SYN → *45-18= 27%*
  (Scan (FILTERED), (CLOSED))
  7.997.748 Pakete für Scan (1.02% der Σ IP)
- 29% FIN ACK (15% Verbindungsabbau)
- **18% SYN ACK (OPEN)**
  (→ 18% Verbindungsaufbau)
- 9% (ACK) RST (Verbindungsreset (Abbau), Scan (OPEN), (CLOSED))
- <1% FIN







Analyse TCP
→ TCP-Verbindungen: Berechnung

- 586.131.923 TCP-Pakete insgesamt
- **5.266.384 SYN-ACK Pakete insgesamt**
  → Indikator für erfolgreiche TCP-Verbindungen
  - **4.326.855 (82,16%) für Well-known Ports**
    - 3.805.706 - Port 80 (HTTP)
    - 234.428 - Port 433 (HTTPS)
    - 170.622 - Port 25 (SMTP)
    - 30.524 - Port 110 (POP3)
    - 806 - Port 119 (NNTP)
    - 611 - Port 179 (BGP)

    mal 7 für TCP-Auf- und Abbau Pakete = 36.864.680
    + 15.085.982 (ACK-RST/+SYN)
- 534.181.261 TCP-Datentransfer Pakete

- **101.43 TCP-Pakete pro TCP-Verbindung**

Analyse TCP
→ TCP-Verbindungen: Port 80 / Port 25

Port 80 (HTTP)

- 378.390.911 TCP-Pakete (Port 80, DST+SRC)
- **3.805.706 SYN-ACK Pakete (Port 80)**
  mal 7 für TCP-Auf- und Abbau Pakete = 26.639.942
  + 10.861.907 (ACK-RST/+SYN – 72% Anteil)
- 340.889.061 TCP-Datentransfer Pakete
- **89,57 TCP-Pakete (Port 80) pro TCP-Verbindung**

Port 25 (SMTP)

- 6.787.219 TCP-Pakete (Port 25, DST+SRC)
- **170.622 SYN-ACK Pakete (Port 25)**
  mal 7 für TCP-Auf- und Abbau Pakete = 1.194.354
  + 270.590 (ACK-RST/+SYN – 1,8% Anteil)
- 5.322.275 TCP-Datentransfer Pakete
- **31,19 TCP-Pakete (Port 25) pro TCP-Verbindung**

Hinweis: Die anderen Anwendungen „streamen“ mehr

Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- **Optimierungen des TCP-Protokolls**
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- Zusammenfassung

Optimierungen des TCP-Protokolls
→ Motivation

- An einer TCP-Verbindung sind immer Instanzen mit begrenzten und endlichen Ressourcen beteiligt.
- Wesentlich sind dabei:
  - vorhandene und benötigte **Datenpuffer**, z.B. Empfangspuffer in den Endsystemen oder Puffer jeglicher Art in Routern,
  - die **Übertragungskapazität** (Bandbreite) und
  - die **Übertragungszuverlässigkeit** (Übertragungsfehler, Paketverluste).
- Allerdings sind diese Werte i.d.R. nicht alle bekannt, noch kann ihre Konstanz vorausgesetzt werden.
- Daher müssen sich die im TCP-Protokoll verwendeten Algorithmen von **selbst an die Systemgegebenheit anpassen**.
- Im folgenden werden einige **Heuristiken** beschrieben, mit denen eine Verbesserung der Nutzrate bei TCP erzielt werden kann.

Optimierungen des TCP-Protokolls
→ Nagle Algorithmus

- Anstatt dem Sender jede kleine Datenmenge sofort nach der Verfügbarkeit von der Applikation zu senden, wird gewartet bis ein **effektiver Versand** möglich ist.
- Dies ist zum einen erreicht, wenn die angesammelten Datenmengen die Größe eines maximalen Segmentes erreicht hat, und zum anderen dann, wenn der Empfang aller vorher gesandten Daten bestätigt ist.

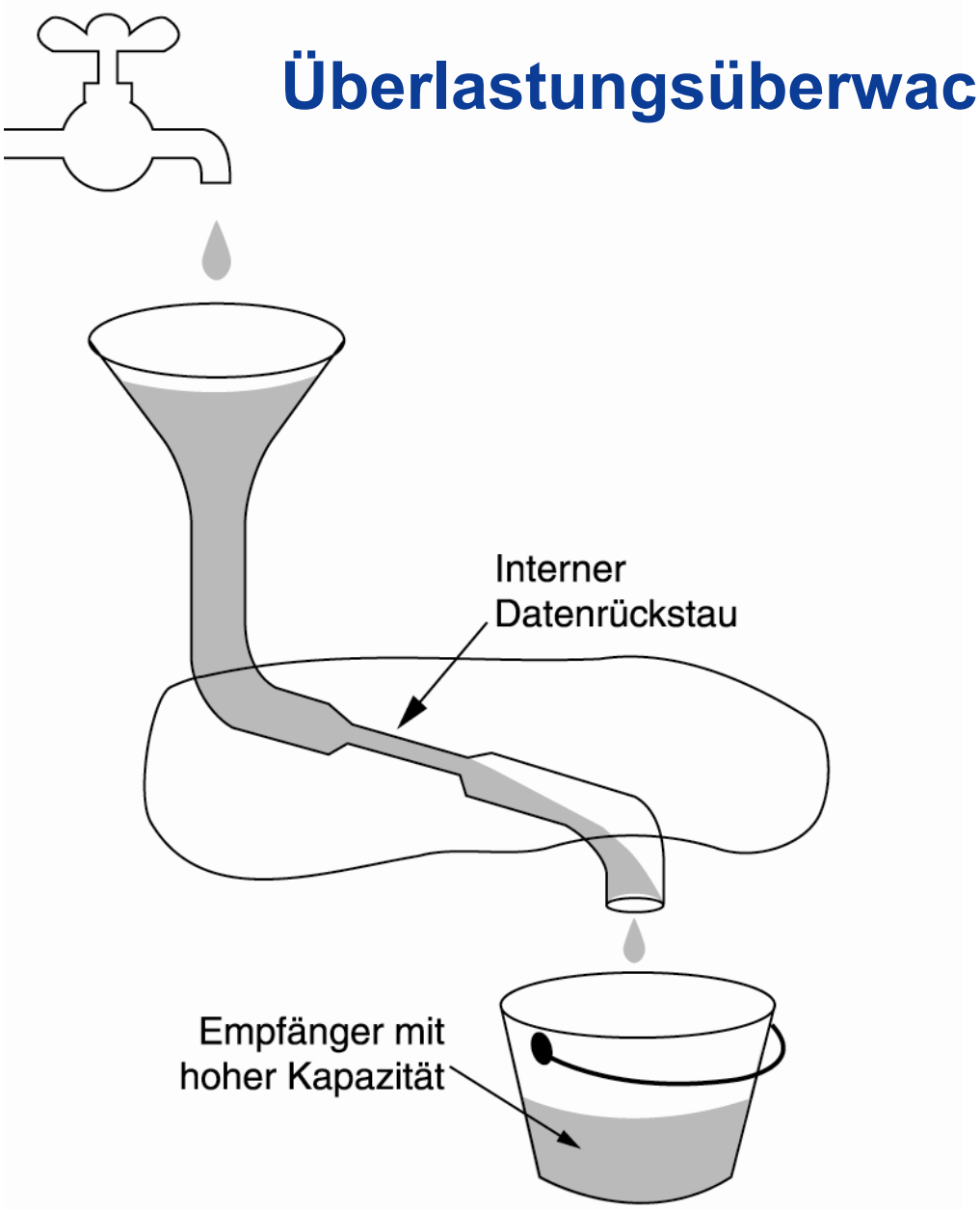
Optimierungen des TCP-Protokolls
→ Delayed Acknowledgement

- Anstatt auf der Empfangsseite Bestätigungen sofort zu senden, nachdem Daten eingetroffen sind, wird noch ein kurzer Zeitraum (bis zu 200ms) gewartet, um eventuelle Synergieeffekte mit anderen zu sendenden Paketen auszunutzen.
- So ist es z.B. möglich, mehrere **Bestätigungen zu einer zusammen zu fassen** (Kumulative Bestätigung - siehe PM FTP).
- Zusätzlich kann die **Bestätigung** im Rahmen eines in Gegenrichtung zu versendenden Datenpakets **mitgesandt** werden (**Huckepack- oder „piggyback“-Mechanismus**).
- Dies spart die Übertragung eines separaten ACK-Paketes (siehe PM).
- Wenn innerhalb dieser Zeit keine Daten in die Gegenrichtung gesendet werden, wird ein reines ACK-Paket von TCP übertragen.
- Der (200 ms)-Timer wird nicht für jedes Paket aufgezogen, sondern läuft global, d.h. alle 200 ms werden ACKs verschickt, die noch offen sind.

Optimierungen des TCP-Protokolls
→ Analogie: Paketverluste im Netz

- In dieser Analogie ist der begrenzende Faktor nicht das Fassungsvermögen des Eimers, sondern die interne Kapazität des Netzes (Rohr).

- Fließt zu viel Wasser zu schnell zu, geht Wasser verloren (in diesem Fall läuft der Trichter über - **Puffer in den Routern**).



Optimierungen des TCP-Protokolls
→ Congestion Windows (1/5) - Idee

- Paketverluste sind hauptsächlich die Folgen von der Überlastung einzelner Knoten (Router; zu wenig Puffer im Router).
- Aus der **Prämisse, Paketverluste** so weit wie möglich zu vermeiden, folgt zwingend die Aufgabe, **Überlaststituationen** zu verhindern.
- Ein dahingehend optimiertes Protokoll muss zum einen eine zuverlässige und zügige Annäherung an eine maximal mögliche Auslastung finden, zum anderen muss es bei gegebener Überlast diesen Zustand so schnell wie möglich entspannen.
- Als Implementierungsmittel setzt TCP auf das Konzept eines zusätzlichen Fensters, dem „Congestion Window“ (Überlastungsfenster), als obere Schranke des möglichen Datenversandes.
- Das „Congestion Window“ wird wie das vom Empfänger mitgeteilte „Flow Control Window“ in Byte geführt, **es besteht keine Notwendigkeit, das „Congestion Window“ zwischen den Partnern zu übertragen.**
- Ein Sender darf immer nur **das Minimum** aus „Congestion Window“ und „Flow Control Window“ an Daten senden.
- Im „Congestion Window“ spiegelt sich die **senderseitige „Wahrnehmung“ der Auslastungssituation** wieder.

Optimierungen des TCP-Protokolls
→ Congestion Windows (2/5) - Allgemein

- Beim Aufbau einer Verbindung initialisiert der Sender das Übertragungsfenster auf die Größe des maximalen Segmentes.
- Damit wird ein maximales Segment gesendet.
- Wird diese Segment bestätigt, bevor der (Retransmission) Timer abläuft, werden die Bytes des Segments dem Überlastungsfenster hinzugefügt, so dass es die Größe zweier Segmente mit maximaler Größe hat, und es werden zwei Segmente gesendet.
- Umfasst das Überlastungsfenster n Segmente und werden alle n Segmente rechtzeitig bestätigt, wird das Übertragungsfenster um die Bytezählung erhöht, die n Segmente entspricht.
- Das bedeutet, das jedes rechtzeitig bestätigte Überlastungsfenster das Überlastungsfenster verdoppelt.
- Das Überlastungsfenster nimmt exponentiell zu, bis entweder ein Timeout auftritt oder das Fenster des Empfängers erreicht ist.
- Wenn Mengen von 1.024, 2.048 und 4.096 Byte korrekt übertragen werden, eine Übertragung von 8.192 Byte aber zu einem Timeout führt, soll das Übertragungsfenster auf 4.096 gesetzt werden, da sonst eine Überlastung entsteht.

Optimierungen des TCP-Protokolls
→ Congestion Windows (3/5) - Allgemein

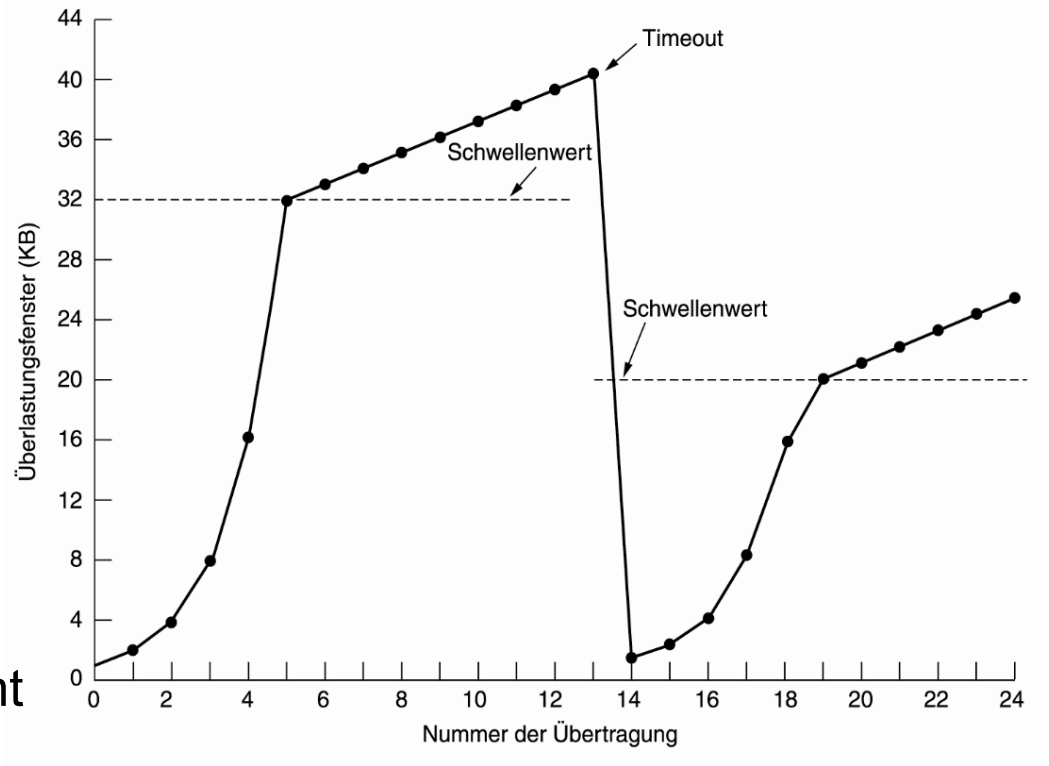
- Solange das Überlastungsfenster auf 4.096 bleibt, werden keine größeren Einzelmengen übertragen, ganz gleich, wie viel Fensterplatz der Empfänger gewählt hat.
- Diesen Algorithmus nennt man ***Slow Start***, er ist aber überhaupt nicht langsam, sondern exponentiell.
- Alle TCP-Implementierungen müssen ihn unterstützen.

Optimierungen des TCP-Protokolls
→ Congestion Windows (4/5) - Internet

- Der im **Internet genutzte „Slow Start“** arbeitet nur mit einem zusätzlichen Parameter, dem **Schwellenwert** (Threshold).
- Dieser Schwellwert beträgt anfangs 64 Kbyte.
- Läuft ein Timer ab, wird der Schwellenwert auf die Hälfte des aktuellen Überlastungsfenster gesetzt, und das Überlastungsfenster wird auf maximales Segment zurückgesetzt.
- Mit „Slow Start“ wird dann bestimmt, was das Netz bewältigen kann, allerdings hört das exponentielle Wachstum auf, wenn der Schwellenwert erreicht ist.
- Ab diesem Punkt vergrößern erfolgreiche Übertragungen das Überlastungsfenster linear (um ein maximales Segment pro „Übertragung“ anstatt eines pro Segment).
- Dieser **Algorithmus „vermutet“**, dass es wahrscheinlich akzeptabel ist, das Überlastungsfenster auf die Hälfte zu reduzieren und sich dann allmählich von dort nach oben zu arbeiten.

Optimierungen des TCP-Protokolls
→ Congestion Windows (5/5) - Internet - Beispiel

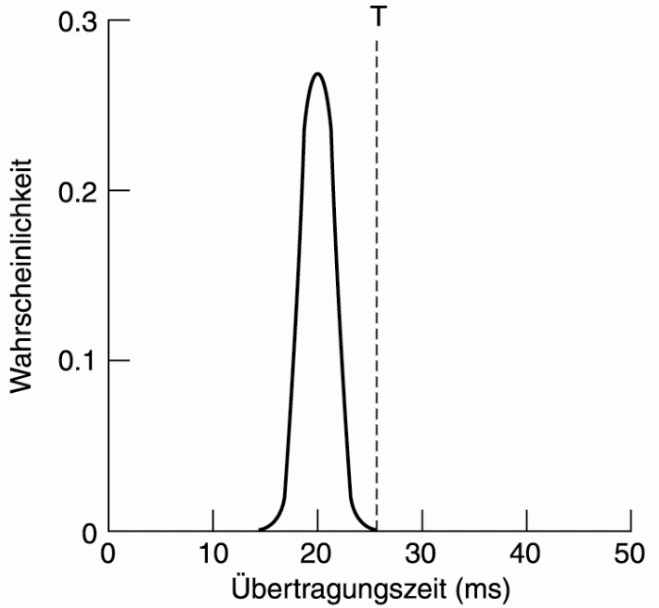
- Die maximale Segmentgröße ist 1.024 Byte.
- Anfangs umfasst das Überlastungsfenster 64 Kbyte, dann findet ein Timeout statt, so dass der Schwellenwert auf 32 Kbyte gesetzt wird (Übertragung 0).
- Das Überlastungsfenster wird auf 1.024 Byte (1KB) gesetzt.
- Dann vergrößert sich das Überlastungsfenster exponentiell, bis der Schwellenwert (32 KByte) erreicht.
- Ab dann nimmt es linear zu.
- Bei der Übertragung 13 kommt es zu einem Timeout.
- Der Schwellenwert wird auf die Hälfte des aktuellen Fenster gesetzt, und „Slow Start“ beginnt vom neuen.
- Kommen die Bestätigungen von Übertragung 14 an, wird das Überlastungsfenster durch die ersten vier jeweils verdoppelt, danach nimmt es wieder linear zu.
- Treten keine Timeouts mehr auf, nimmt das Überlastungsfenster bis auf die Größe des Empfängerfenster zu.



Optimierungen des TCP-Protokolls
→ Verwaltung von Timer in TCP (1/5)

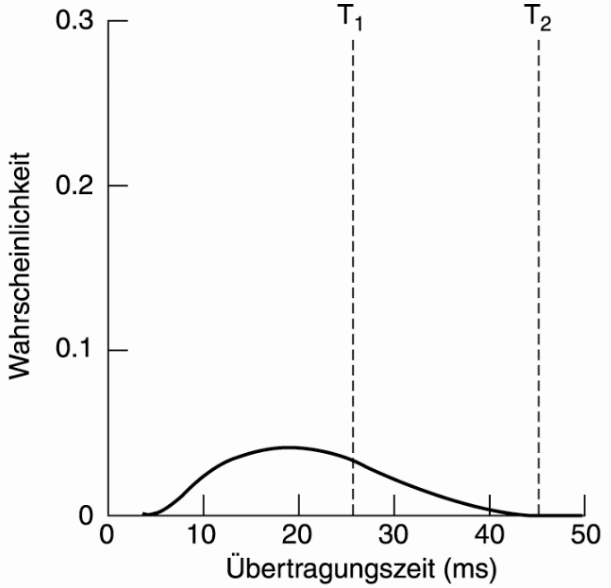
- Der wichtigste Timer bei TCP ist der Retransmission-Timer (Timer für die erneute Übertragung).
- Wird ein Segment gesendet, startet der Retransmission-Timer.
- Wird das Segment bestätigt, bevor der Timer abläuft, wird der Timer gestoppt.
- Läuft anderseits der Timer vor der Bestätigung ab, wird das Segment erneut übertragen (und der Timer startet wieder von vorn).
- **Die Frage ist nun:**
  Wie lang soll die Zeitspanne bis zum Timeout sein?

- Bei Timern auf der Sicherungsschicht ist die Übertragungszeit gut vorhersehbar (d.h. sie unterliegt nur geringen Schwankungen).
- Daher kann der Timer so gesetzt werden, dass er abläuft, kurz nachdem eine Bestätigung erwartet wird.



Optimierungen des TCP-Protokolls
→ Verwaltung von Timer in TCP (2/5)

- TCP muss sich mit einer anderen Umgebung befassen.
- Die Wahrscheinlichkeitsdichte-Funktion der Zeit, die es dauert, bis eine TCP-Bestätigung zurückkommt, sieht anders aus!
- Die Bestimmung der Übertragungszeit bis zum Ziel ist nicht einfach.



- Ist der **Timeout zu kurz**, z.B. T_1, erfolgen unnötige Neuübertragungen und das Internet wird mit **nutzlosen Paketen überfrachtet**.
- Ist die **Zeitspanne zu lang** (T_2), entstehen Leistungseinbußen durch **lange Übertragungsverzögerungen** für Neuübertragung.
- Des Weiteren können sich der Mittelwert und die Varianz der Verteilung der Ankunftszeit von Bestätigungen schnell innerhalb von ein paar Sekunden ändern, wenn Überlastung entsteht oder aufgelöst wird.
- Eine Lösung dieser Aufgabenstellung ist ein sehr **dynamischer Algorithmus**, der das **Timeout-Intervall** auf der Grundlage laufender Messungen der Netzleistung fortlaufend anpasst.

Optimierungen des TCP-Protokolls
→ Verwaltung von Timer in TCP (3/5)

- Der bei TCP allgemein verwendete Algorithmus ist auf Jacobson (1988) zurückzuführen und funktioniert wie folgt:
  - Für jede Verbindung verwaltet TCP die **Variable *RTT (Round-Trip-Timer)***.
  - Das ist die aktuell **beste Schätzung der Übertragungszeit** zum fraglichen Ziel bis zum Erhalt der Bestätigung.
  - Wird ein Segment gesendet, startet ein Timer.
  - Er überwacht einerseits, wie lange die Bestätigung braucht, und löst anderseits eine Neuübertragung aus, wenn dies zu lange dauert.
  - Wenn die Bestätigung zurückkommt, bevor der Timer abläuft, misst TCP, wie lange die Bestätigung unterwegs war, z.B. die **Zeit „M“**.

Optimierungen des TCP-Protokolls
→ Verwaltung von Timer in TCP (4/5)

- ***RTT** wird* unter Verwendung der folgende Formel neu berechnet:
- **RTT = a * *RTT* + (1 - a) * M**

- Wobei „a“ ein Glättungsfaktor ist, der festlegt, wie stark der alte Wert gewichtet wird (i.d.R. ist a = 7/8 = 0,875).
- Beispiel-Rechnung:

alt RTT	M	neu RTT
200	50	181,25
181,25	50	164,84
164,84	50	150,48
150,48	50	137,92
...	...	...
50	50	50
50	200	68,75

Optimierungen des TCP-Protokolls
→ Verwaltung von Timer in TCP (5/5)

- Der Wert für den Timeout = 2 * *RTT*
- Es gibt aber auch eine dynamische Berechnung.

Timout = *RTT* + 4*D

(D = die Berechnung der Abweichung)

Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- **Drahtlose TCP Verbindungen**
- Protokollmitschnitte
- Zusammenfassung

Drahtlose TCP Verbindungen
→ Problembeschreibung (1/2)

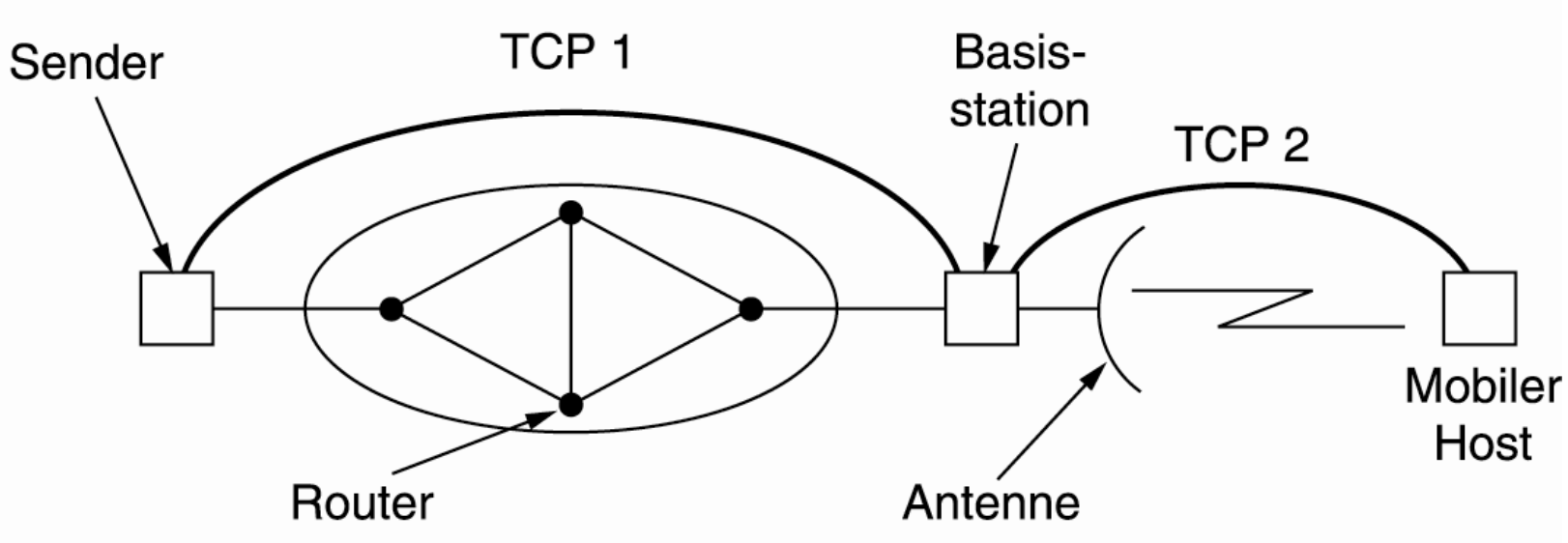
- Theoretisch sollten Transportprotokolle von der Technologie der zugrunde liegenden Netzschicht unabhängig sein.
- Insbesondere sollte es für TCP gleichgültig sein, ob IP über Glasfaser oder Funk läuft.
- In der Praxis spielt das aber sehr wohl eine Rolle, weil die meisten TCP-Implementierungen sorgfältig auf der Grundlage von Annahmen optimiert wurden, die auf Kabelnetze zutreffen, bei drahtlosen Netzen aber versagen.
- Wenn man die Eigenschaften der drahtlosen Übertragung ignoriert, ist eine TCP-Implementierung unter Umständen zwar logisch korrekt, hat aber eine schwache Leistung.
- **Das prinzipielle Problem ist der Algorithmus für die Überlastungsüberwachung.**
- Fast alle TCP-Implementierungen basieren heute auf der Annahme, dass **Timeouts durch Überlastung** (durch Paketverlust) oder Übertragungsfehler verursacht werden.

Drahtlose TCP Verbindungen
→ Problembeschreibung (2/2)

- Läuft ein Timer ab, verlangsamt sich TCP (z.B. durch Slow-Start).
- Mit diesem Ansatz wird bezweckt, die Netzbelastung zu reduzieren, um Überlastungen zu mindern.
- Leider sind drahtlose Übertragungsleitungen höchst unzuverlässig.
- Sie verlieren die ganze Zeit Pakete.
- Der richtige Ansatz ist hier, verlorene Pakete so schnell wie möglich erneut zu übertragen.
- Durch die Verlangsamung der Übertragung verschlimmert sich die Situation noch.
  - In **einem Kabelnetz sollte der Sender langsamer senden**, wenn ein Paket verloren geht.
  - In **einem drahtlosen Netz muss der Sender die Pakete schneller wiederholen!**
- Weiß der Sender nicht, über welches Netz er überträgt, kann kaum eine richtige Entscheidung getroffen werden.

Drahtlose TCP Verbindungen
→ Eine mögliche Lösung (1/2)

- Eine mögliche Lösung ist das **indirekte TCP**.
- Dabei wird die TCP-Verbindung in zwei **getrennte Verbindungen** aufgeteilt.



- Die erste Verbindung läuft vom Sender zur Basisstation; die zweite führt von der Basisstation zum Empfänger.
- Die Basisstation kopiert lediglich Pakete zwischen den Verbindungen.

Drahtlose TCP Verbindungen
→ Eine mögliche Lösung (2/2)

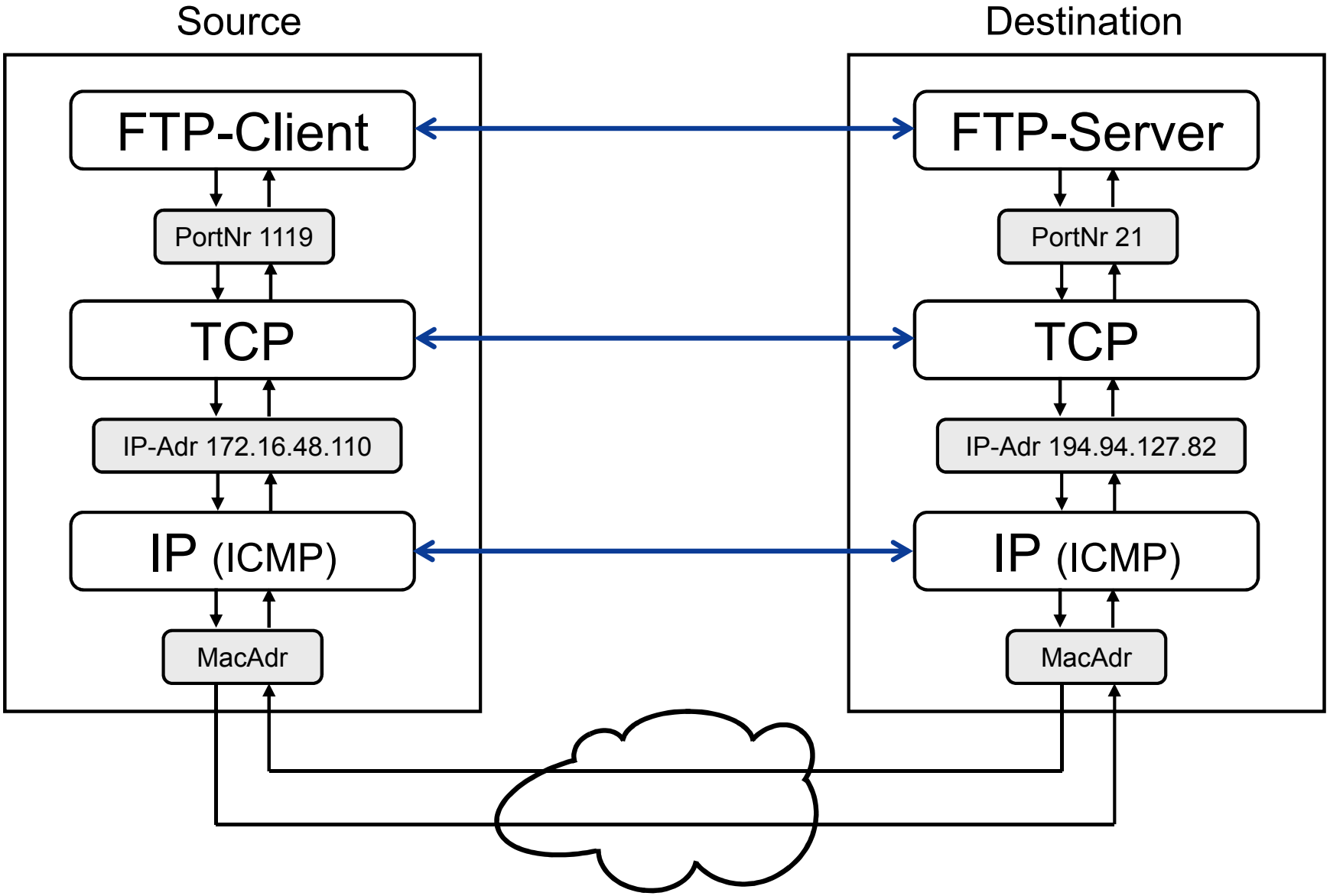
- **Vorteile:**
  - Beide Verbindungen sind nun homogen.
  - Timeouts auf der ersten Verbindung können den Sender verlangsamen, während er durch Timouts auf der zweiten Verbindung beschleunigt werden kann.
  - Darüberhinaus lassen sich auch weitere Parameter für die beiden Verbindungen getrennt einstellen.

- **Nachteile:**
  - Die TCP-Semantik wird verletzt.
  - Da jeder Teil der Verbindung eine volle TCP-Verbindung ist, bestätigt die Basisstation jedes TCP-Segment auf die übliche Weise.
  - Nur jetzt bedeutet der Empfang einer Bestätigung beim Sender nicht, dass der Empfänger das Segment erhalten hat.
  - Vielmehr hat es die Basisstation.

Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- **Protokollmitschnitte**
- Zusammenfassung

TCP - Transmission Control Protocol
→ Beispiel: FTP



TCP - Transmission Control Protocol
→ Beispiel

Log einer TCP - Verbindung (Übersicht)

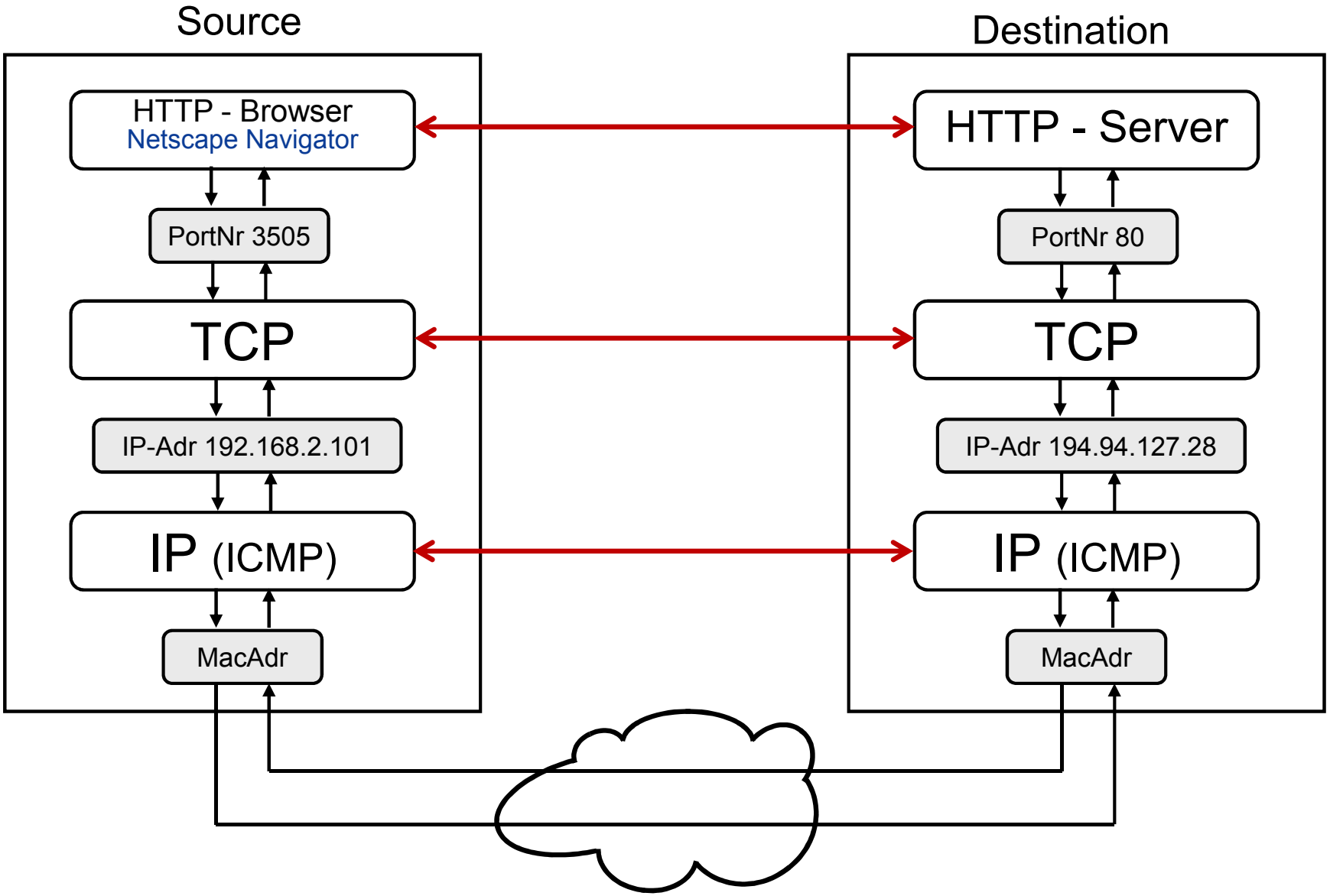
Verbindungsaufbau

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.48.110	194.94.127.82	TCP	1119 > ftp [SYN] Seq=2244048930 Ack=0 Win=64240 Len=0
2	0.003113	194.94.127.82	172.16.48.110	TCP	ftp > 1119 [SYN, ACK] Seq=1439104370 Ack=2244048931 Win=8760 Len=0
3	0.003173	172.16.48.110	194.94.127.82	TCP	1119 > ftp [ACK] Seq=2244048931 Ack=1439104371 Win=64240 Len=0
4	0.142964	194.94.127.82	172.16.48.110	FTP	Response: 220 sun2 FTP server (UNIX(r) System V Release 4.0) ready.
5	0.259567	172.16.48.110	194.94.127.82	TCP	1119 > ftp [ACK] Seq=2244048931 Ack=1439104430 Win=64181 Len=0
6	4.081460	172.16.48.110	194.94.127.82	FTP	Request: USER anonymous
7	4.094896	194.94.127.82	172.16.48.110	FTP	Response: 331 Guest login ok, send ident as password.
8	4.220474	172.16.48.110	194.94.127.82	TCP	1119 > ftp [ACK] Seq=2244048947 Ack=1439104475 Win=64136 Len=0
9	10.153567	172.16.48.110	194.94.127.82	FTP	Request: PASS a@b.c.d
10	10.168716	194.94.127.82	172.16.48.110	FTP	Response: 230 Guest login ok, access restrictions apply.
11	10.314105	172.16.48.110	194.94.127.82	TCP	1119 > ftp [ACK] Seq=2244048961 Ack=1439104523 Win=64088 Len=0
12	13.402383	172.16.48.110	194.94.127.82	FTP	Request: QUIT
13	13.404917	194.94.127.82	172.16.48.110	FTP	Response: 221 Goodbye.
14	13.405427	172.16.48.110	194.94.127.82	TCP	1119 > ftp [FIN, ACK] Seq=2244048967 Ack=1439104537 Win=64074 Len=0
15	13.407371	194.94.127.82	172.16.48.110	TCP	ftp > 1119 [ACK] Seq=1439104537 Ack=2244048968 Win=8760 Len=0
16	13.407427	194.94.127.82	172.16.48.110	TCP	ftp > 1119 [FIN, ACK] Seq=1439104537 Ack=2244048968 Win=8760 Len=0
17	13.407453	172.16.48.110	194.94.127.82	TCP	1119 > ftp [ACK] Seq=2244048968 Ack=1439104538 Win=64074 Len=0

Verbindungsabbau

TCP - Transmission Control Protocol

→ Beispiel - Teilausschnitt aus einer HTTP-Session



Siehe auch HTTP-Vorlesung

TCP - Transmission Control Protocol

→ Beispiel - Teilausschnitt aus einer HTTP-Session

No.	Time	Source	Destination	Protocol	Info
134	3.175168	192.168.2.101	194.94.127.28	TCP	3605 > http [SYN] Seq=61003783 Ack=0 Win=64240 Len=0
148	3.284466	194.94.127.28	192.168.2.101	TCP	http > 3605 [SYN, ACK] Seq=3740781940 Ack=61003784 Win=15466 Len=0
149	3.284510	192.168.2.101	194.94.127.28	TCP	3605 > http [ACK] Seq=61003784 Ack=3740781941 Win=64676 Len=0
150	3.284857	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
160	3.429339	194.94.127.28	192.168.2.101	TCP	http > 3605 [ACK] Seq=3740781941 Ack=61004233 Win=15466 Len=0
161	3.434794	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
162	3.459313	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sl.gif HTTP/1.0
168	3.564831	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
171	3.616456	192.168.2.101	194.94.127.28	TCP	3605 > http [FIN, ACK] Seq=61004670 Ack=3740782663 Win=63954 Len=0
174	3.693432	194.94.127.28	192.168.2.101	TCP	http > 3605 [ACK] Seq=3740782663 Ack=61004671 Win=15466 Len=0
175	3.694331	194.94.127.28	192.168.2.101	TCP	http > 3605 [FIN, ACK] Seq=3740782663 Ack=61004671 Win=15466 Len=0
176	3.694354	192.168.2.101	194.94.127.28	TCP	3605 > http [ACK] Seq=61004671 Ack=3740782664 Win=63954 Len=0

TCP - Transmission Control Protocol

→ Beispiel - Verbindungsaufbau

Nr. 134 Transmission Control Protocol, Src Port: 3606 (3606), Dst Port: http (80), Seq: 61094067, Ack: 0, Len: 0

Source port: 3606 (3606)
Destination port: http (80)
Sequence number: 61094067
Header length: 28 bytes
Flags: 0x0002 (SYN)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...0 .... = Acknowledgment: Not set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..1. = Syn: Set
....0 = Fin: Not set
Window size: 64240
Checksum: 0x38e9 (correct)
Options: (8 bytes)
Maximum segment size: 1460 bytes
NOP
NOP
SACK permitted

Nr. 148 Transmission Control Protocol, Src Port: http (80), Dst Port: 3606 (3606), Seq: 3740935862, Ack: 61094068, Len: 0

Source port: http (80)
Destination port: 3606 (3606)
Sequence number: 3740935862
Acknowledgement number: 61094068
Header length: 28 bytes
Flags: 0x0012 (SYN, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..1. = Syn: Set
....0 = Fin: Not set
Window size: 15466
Checksum: 0xf1e3 (correct)
Options: (8 bytes)
Maximum segment size: 1406 bytes
NOP
NOP
SACK permitted

Nr. 149 Transmission Control Protocol, Src Port: 3606 (3606), Dst Port: http (80), Seq: 61094068, Ack: 3740935863, Len: 0

Source port: 3606 (3606)
Destination port: http (80)
Sequence number: 61094068
Acknowledgement number: 3740935863
Header length: 20 bytes
Flags: 0x0010 (ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 64676
Checksum: 0x5e37 (correct)

TCP - Transmission Control Protocol
→ Beispiel - Teilausschnitt aus einer HTTP-Session

Nr. 150 Transmission Control Protocol, Src Port: 3606 (3606), Dst Port: http (80), Seq: 61094068, Ack: 3740935863, Len: 437
Source port: 3606 (3606)
Destination port: http (80)
Sequence number: 61094068
Next sequence number: 61094505
Acknowledgement number: 3740935863
Header length: 20 bytes
Flags: 0x0018 (PSH, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 64676
Checksum: 0xc35c (correct)

- TCP - Datenpaket (HTTP-Request auf Port 80)

Nr. 160 Transmission Control Protocol, Src Port: http (80), Dst Port: 3606 (3606), Seq: 3740935863, Ack: 61094505, Len: 0
Source port: http (80)
Destination port: 3606 (3606)
Sequence number: 3740935863
Acknowledgement number: 61094505
Header length: 20 bytes
Flags: 0x0010 (ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 15466
Checksum: 0x1cbd (correct)

- TCP ACK-Paket (Bestätigung von Nr. 150)
- für 437 Byte
- nach 144 ms

TCP - Transmission Control Protocol
→ Beispiel - Teilausschnitt aus einer HTTP-Session

Nr. 161 Transmission Control Protocol, Src Port: http (80), Dst Port: 3606 (3606), Seq: 3740935863, Ack: 61094505, Len: 340

Source port: http (80)
Destination port: 3606 (3606)
Sequence number: 3740935863
Next sequence number: 3740936203
Acknowledgement number: 61094505
Header length: 20 bytes
Flags: 0x0018 (PSH, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 15466
Checksum: 0xcf67 (correct)

- TCP - Datenpaket (HTTP-Response auf Port 3606)

Nr. 162 Transmission Control Protocol, Src Port: 3606 (3606), Dst Port: http (80), Seq: 61094505, Ack: 3740936203, Len: 437

Source port: 3606 (3606)
Destination port: http (80)
Sequence number: 61094505
Next sequence number: 61094942
Acknowledgement number: 3740936203
Header length: 20 bytes
Flags: 0x0018 (PSH, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 64336
Checksum: 0xbba7 (correct)

- TCP - Datenpaket (HTTP-Request auf Port 80)
- mit Piggyback-ACK (Bestätigung von Nr. 161)
- für 340 Byte
- nach 25 ms

TCP - Transmission Control Protocol

→ Beispiel - Teilausschnitt aus einer HTTP-Session

Nr. 168 Transmission Control Protocol, Src Port: http (80), Dst Port: 3606 (3606), Seq: 3740936203, Ack: 61094942, Len: 339

Source port: http (80)
Destination port: 3606 (3606)
Sequence number: 3740936203
Next sequence number: 3740936542
Acknowledgement number: 61094942
Header length: 20 bytes
Flags: 0x0018 (PSH, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 15466
Checksum: 0x9ee5 (correct)

- TCP - Datenpaket (HTTP- Response auf Port 3606)
- mit Piggyback-ACK (Bestätigung von Nr. 162)
- für 437 Byte
- nach 105 ms

TCP - Transmission Control Protocol
→ Beispiel - Verbindungsabbau - Client-Server

Nr. 171 Transmission Control Protocol, Src Port: 3606 (3606), Dst Port: http (80), Seq: 61094942, Ack: 3740936542, Len: 0

Source port: 3606 (3606)
Destination port: http (80)
Sequence number: 61094942
Acknowledgement number: 3740936542
Header length: 20 bytes
Flags: 0x0011 (FIN, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....1 = Fin: Set
Window size: 63997
Checksum: 0x5acc (correct)

- TCP - Abbau-Request
- mit Piggyback-ACK (Bestätigung von Nr. 168)
- für 340 Byte
- nach 52 ms

Nr. 174 Transmission Control Protocol, Src Port: http (80), Dst Port: 3606 (3606), Seq: 3740936542, Ack: 61094943, Len: 0

Source port: http (80)
Destination port: 3606 (3606)
Sequence number: 3740936542
Acknowledgement number: 61094943
Header length: 20 bytes
Flags: 0x0010 (ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 15466
Checksum: 0x1860 (correct)

TCP - Transmission Control Protocol

→ Beispiel - Verbindungsabbau - Server-Client

Nr. 175 Transmission Control Protocol, Src Port: http (80), Dst Port: 3606 (3606), Seq: 3740936542, Ack: 61094943, Len: 0

Source port: http (80)
Destination port: 3606 (3606)
Sequence number: 3740936542
Acknowledgement number: 61094943
Header length: 20 bytes

Flags: 0x0011 (FIN, ACK)

0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....1 = Fin: Set

Window size: 15466
Checksum: 0x185f (correct)

Nr. 176 Transmission Control Protocol, Src Port: 3606 (3606), Dst Port: http (80), Seq: 61094943, Ack: 3740936543, Len: 0

Source port: 3606 (3606)
Destination port: http (80)
Sequence number: 61094943
Acknowledgement number: 3740936543
Header length: 20 bytes

Flags: 0x0010 (ACK)

0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set

Window size: 63997
Checksum: 0x5acb (correct)

TCP - Transmission Control Protocol
→ Beispiel - Zusammenfassung

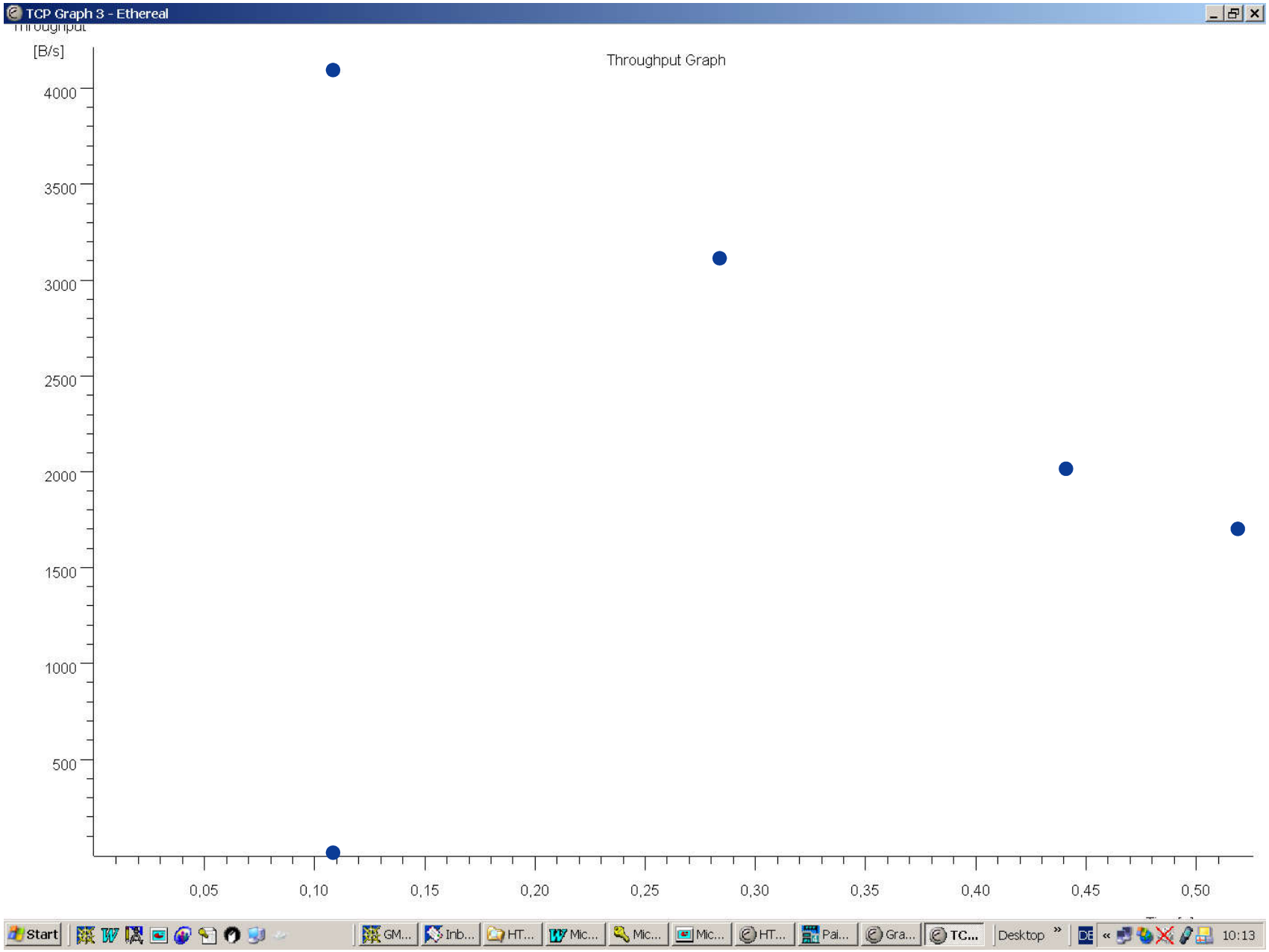
- Die Segmentgröße ist beim Client 1460 Byte und beim Server 1406 Byte (siehe Optionen beim Verbindungsaufbau)

- Zeiten
  - Verbindungsaufbau 109 ms
  - Datentransfer 280 ms
  - Verbindungsabbau 130 ms
  - Summe 519 ms

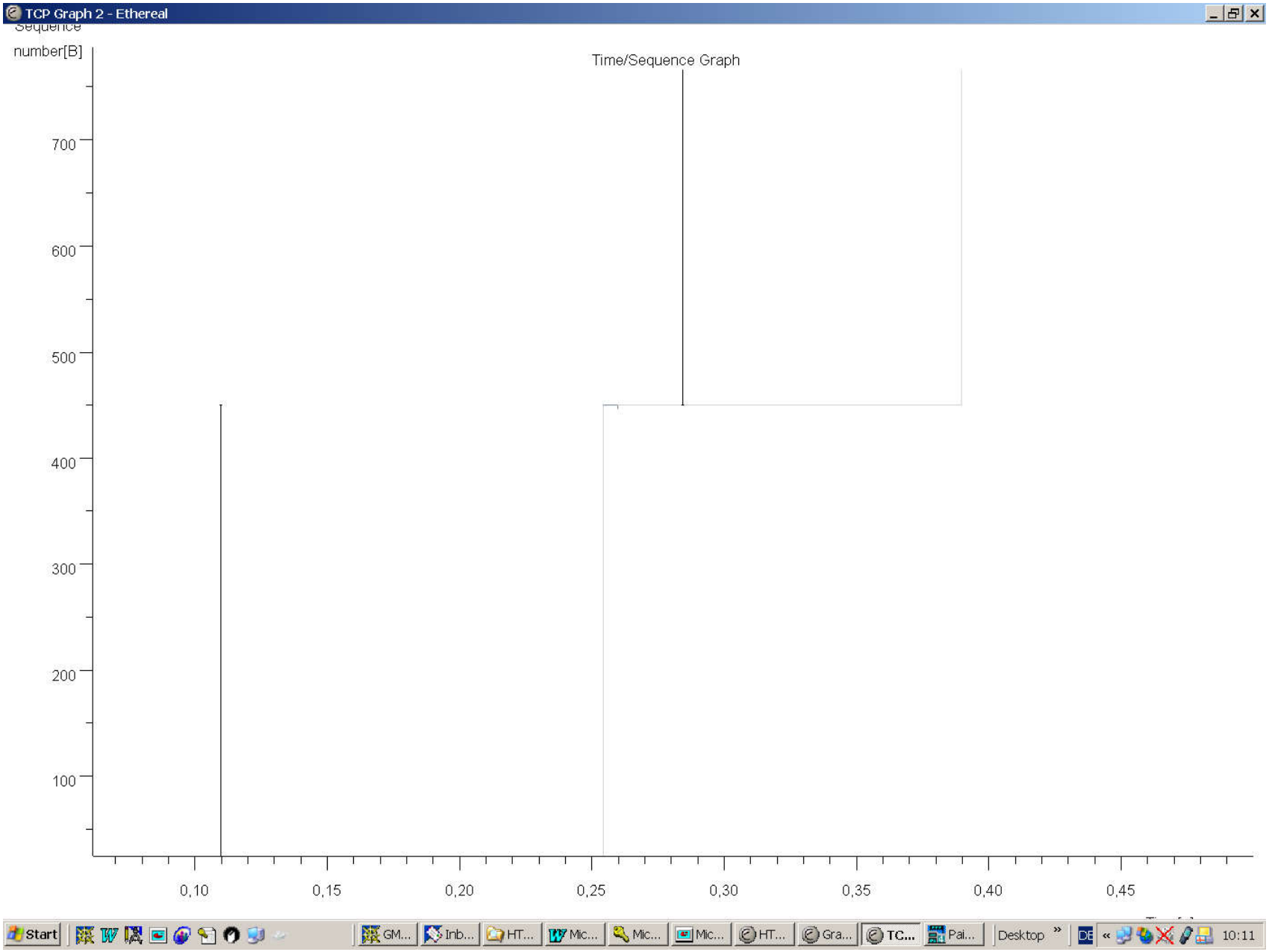
- WINDOW SIZE
  - Client (15466) immer gleich
  - Server (64240) schwankt etwas (-340 Byte)

- Die HTTP-Applikation nutzt den Push-Mechanismus (PSH)!

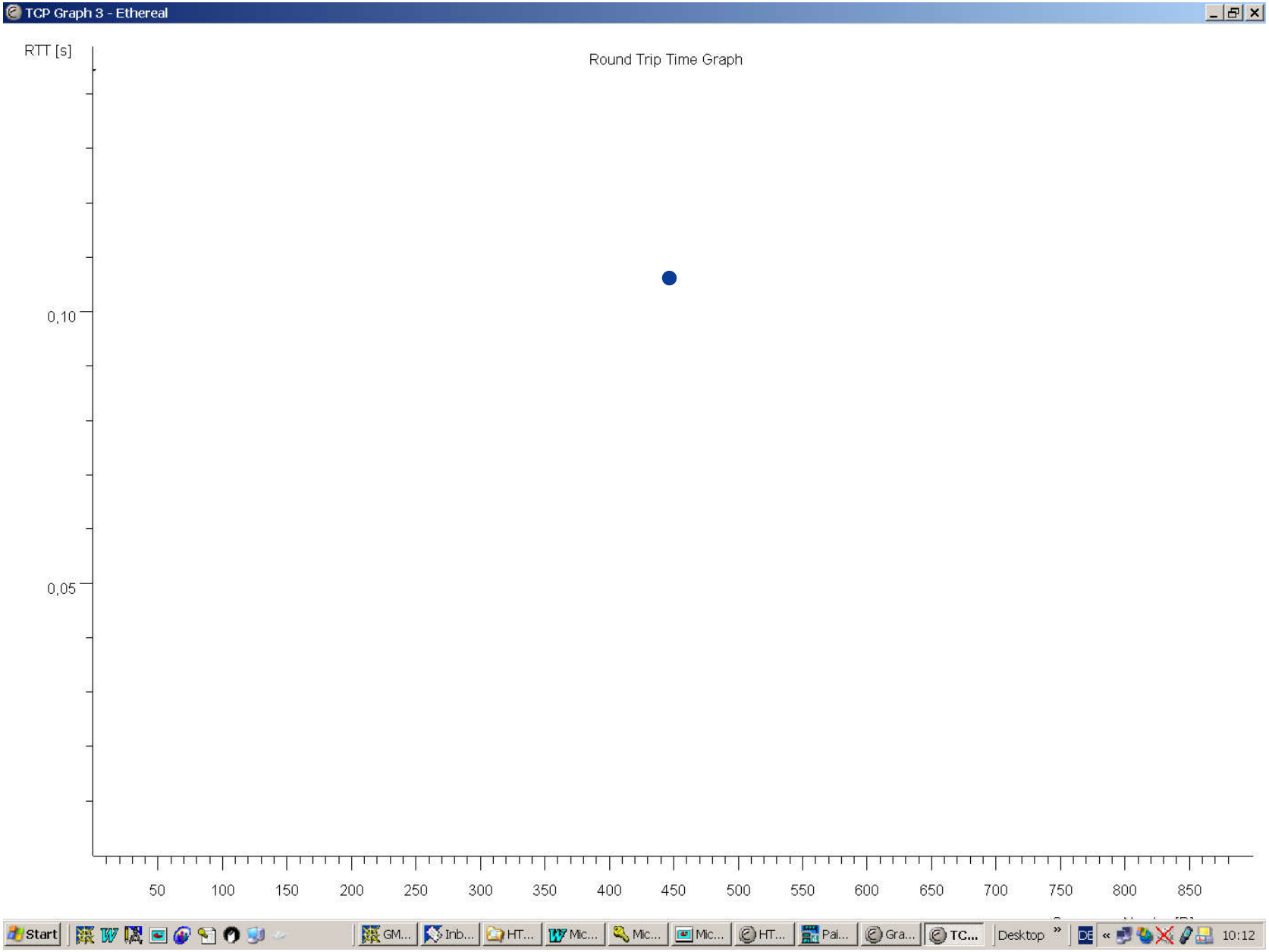
TCP - Transmission Control Protocol
→ Beispiel - Durchsatz



TCP - Transmission Control Protocol
→ Beispiel - Sequence



TCP - Transmission Control Protocol
→ Beispiel - RTT



TCP - Transmission Control Protocol

→ Beispiel - Wiederholung eines Paketes (FTP-Session)

```
No. Time        Source          Destination     Protocol Info
273 79.574201   192.168.2.101   194.94.127.28   FTP-DATA FTP Data: 1406 bytes

274 79.574232   192.168.2.101   194.94.127.28   FTP-DATA FTP Data: 54 bytes
Transmission Control Protocol, Src Port: 3490 (3490), Dst Port: 1885 (1885), Seq: 1073769780, Ack: 708235640, Len: 54
   Sequence number: 1073769780
   Next sequence number: 1073769834
   Acknowledgement number: 708235640


275 79.667553   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073762534 Win=15466
276 79.724329   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073763994 Win=15466

277 79.724405   192.168.2.101   194.94.127.28   FTP-DATA FTP Data: 182 bytes
Transmission Control Protocol, Src Port: 3490 (3490), Dst Port: 1885 (1885), Seq: 1073769834, Ack: 708235640, Len: 182
   Sequence number: 1073769834
   Next sequence number: 1073770016
   Acknowledgement number: 708235640
```

Der Client Versucht sofort abzubauen

```
278 79.767332   192.168.2.101   194.94.127.28   TCP      3490 > 1885 [FIN, ACK] Seq=1073770016 Ack=708235640 Win=64676

279 79.804848   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073765454 Win=15466 Len=0
280 79.874153   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073766914 Win=15466 Len=0
281 79.954217   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073768374 Win=15466 Len=0
282 80.023967   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073769834 Win=15466
283 80.029802   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073769834 Win=15466

284 81.790135   192.168.2.101   194.94.127.28   FTP-DATA FTP Data: 182 bytes (Wiederholung des Pakets)
```

FIN=1

```
285 81.878566   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [ACK] Seq=708235640 Ack=1073770017 Win=15466
```

Wiederholung erst nach ca. 2s; durch das gesetzte FIN-Bit wird der Abbau durch den Client eingeleitet

```
286 81.879903   194.94.127.28   192.168.2.101   TCP      1885 > 3490 [FIN, ACK] Seq=708235640 Ack=1073770017 Win=15466
287 81.879939   192.168.2.101   194.94.127.28   TCP      3490 > 1885 [ACK] Seq=1073770017 Ack=708235641 Win=64676
```

Abbau des Servers

Inhalt

- Ziele und Einordnung
- Adressierung auf der Transportebene
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- Optimierungen des TCP-Protokolls
- Drahtlose TCP Verbindungen
- Protokollmitschnitte
- **Zusammenfassung**

TCP - Transmission Control Protocol
→ Zusammenfassung - Allgemein

- UDP und TCP sind Transportprotokolle
- TCP und UDP sind Ende-zu-Ende Protokolle.
- Mit Hilfe der **Protokoll-Ports**, der Transportadresse, können Kommunikationsanwendungen (HTTP, FTP, ...) adressiert werden.

TCP - Transmission Control Protocol
→ Zusammenfassung - UDP

- **UDP:**
  - Ist ein unzuverlässiger, verbindungsloser Datagramm-Mechanismus.
  - Die Anwendung muss selbst für die gewünschte Zuverlässigkeit sorgen.

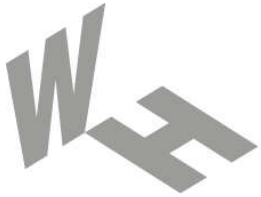
- **Merkmale von UDP:**
  - keine Kontrollmechanismen
  - keine Reihenfolgeüberwachung der Pakete
  - keine Sicherung einer erfolgreichen Übertragung
  - keine Fehlerbehebung
  - keine Zeitüberwachung
  - **zustandslos**
  - **minimaler Protokollmechanismus; wenig Overhead; sehr effektiv**
  - **Einhaltung der Nachrichtengrenzen wird garantiert**
  - **Multi-/Broadcasting ist möglich**

TCP - Transmission Control Protocol
→ Zusammenfassung - TCP - Allgemein

- **TCP:**
  - Ist ein verbindungsorientiertes, zuverlässiges Transportprotokoll.
  - Sorgt für Fehlerbehandlung und stellt die Reihenfolgerichtigkeit her.
  - **Führt eine dynamische Anpassung an unterschiedliche Netzwerk-Merkmale durch!**

TCP - Transmission Control Protocol
→ Zusammenfassung - TCP - Mechanismen

- **Verbindungsorientiert** (Verbindungsaufbau, Datentransfer, Verbindungsabbau)
- Kommunikationspartner sind über **virtuelle, vollduplex-fähige** und **bidirektionale** Leitungen verbunden
- Fehlererkennung durch: **Sequenz-Nr., Bestätigungs-Nr. (**Quittung), **Prüfsumme** (Checksum) und **Zeitüberwachung** (Timer)
- **Segmentwiederholung** bei Datenverlust durch **acknowledgement with retransmission**
- **Segmentierung der Benutzerdaten** (Anwenderprogramme erzeugen nur Datenstrom, ohne Strukturierung in Segmente bzw. Pakete)
- Durch den **Stream-Service** bekommt der Empfänger die Daten in genau der derselben Reihenfolge.
- **Flusskontrolle** zwischen den Endpunkten durch **Sliding-Windows Technik** (Empfänger wird nicht mit Daten überflutet)
- **Überlastüberwachung** des Netzes durch **Slow-Start, Congestion Windows** und **RTT**
- Prioritätssteuerung (**PHS** (push) und **URG** (Interrupt))
- Weitere Optimierung durch **Delayed acknowledgement und Nagle Algorithmus**



Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Die Transportebene

Vielen Dank für Ihre Aufmerksamkeit
Fragen ?

Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

if(is) internet-sicherheit.