Chapter 9

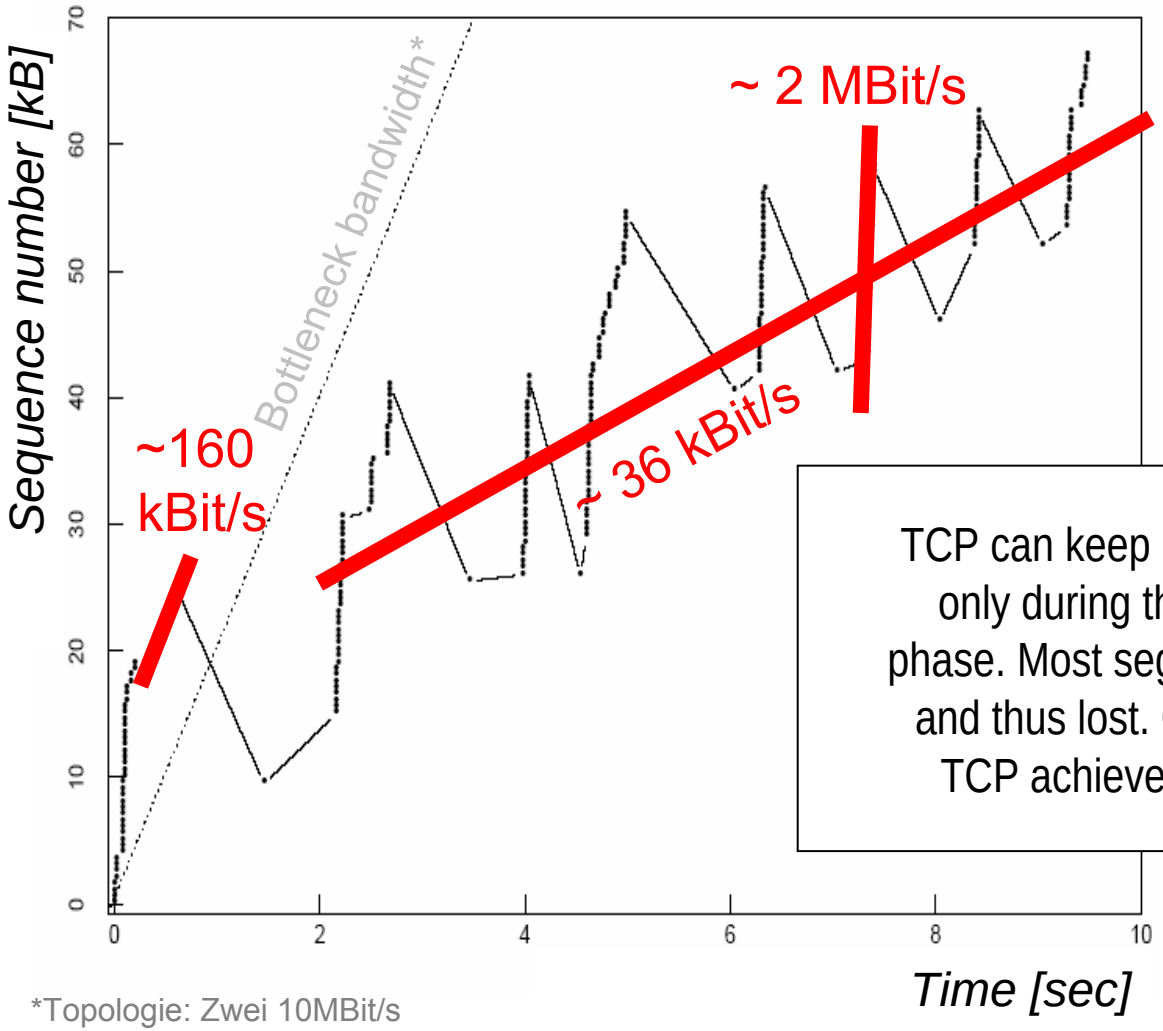
Advanced computer networking

Internet Protocols

Thomas Fuhrmann

Network Architectures
Computer Science Department
Technical University Munich

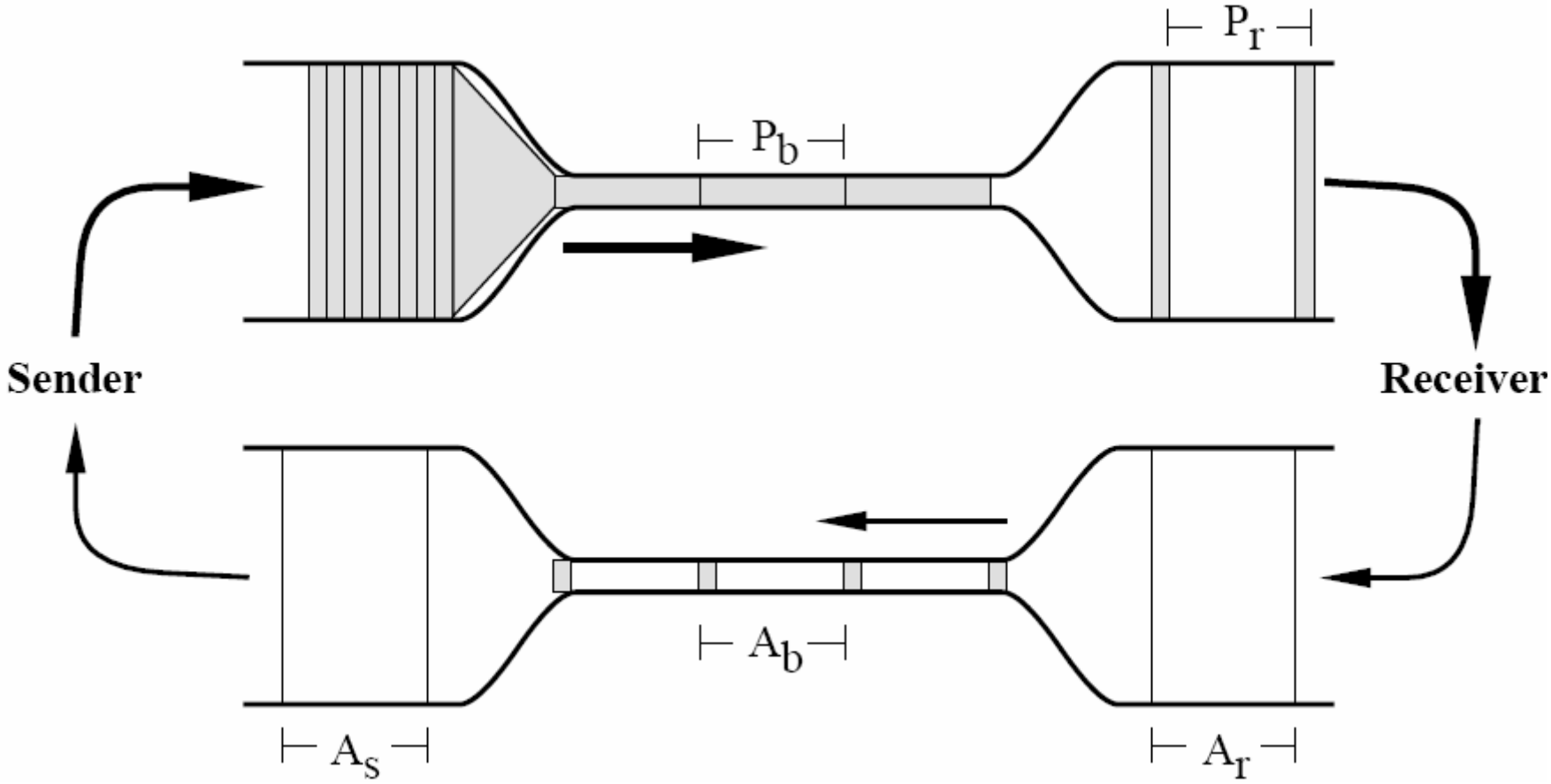
Old „Go-back-N TCP“ Does Not Perform



TCP can keep pace with the bottleneck only during the brief *Self-Clocking* phase. Most segments are sent in bursts and thus lost. Overall such a plain old TCP achieves only 23% efficiency.

*Topologie: Zwei 10MBit/s Ethernet-Subnetze mit 0.16MBit/s Mikrowellen-Link verbunden

TCP Self-Clocking



Quelle: Van Jacobson, Michael J. Karels. *Congestion Avoidance and Control*. Proceedings of SIGCOMM'88, Stanford, CA, August 1988.

TCP Congestion Control

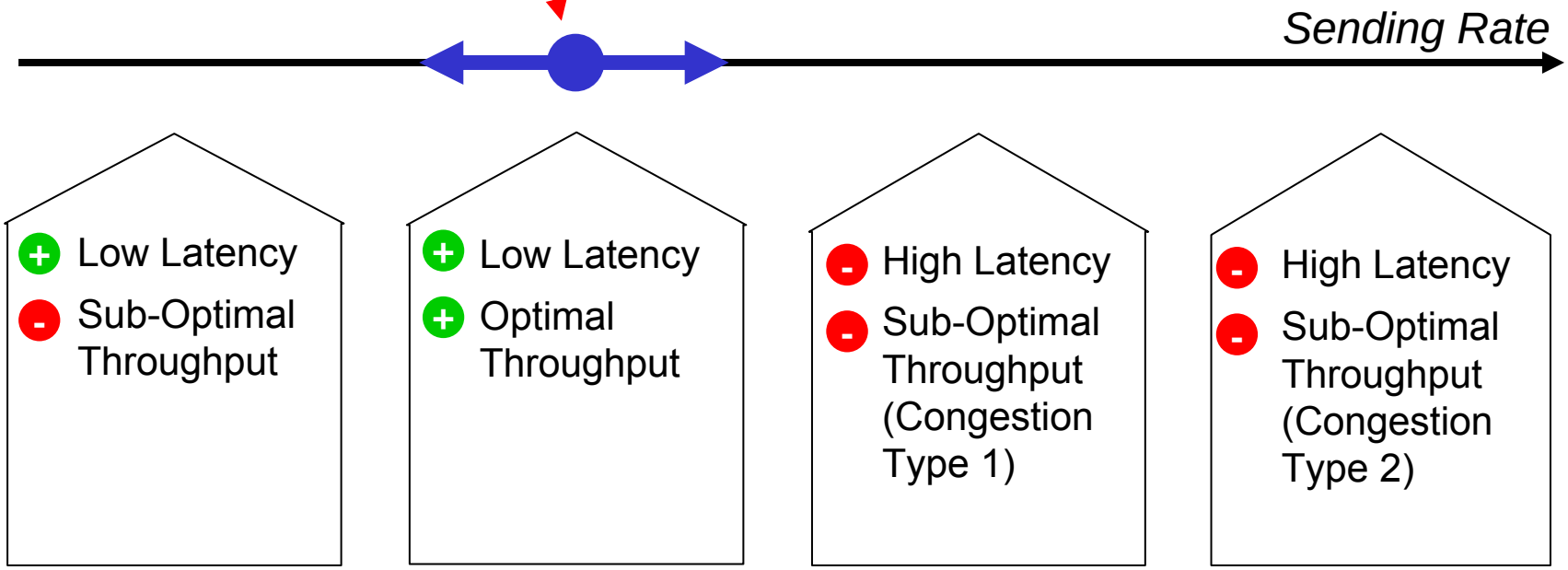
Underutilization:

TCP does not send at the maximum possible rate.

The ideal sending rate is determined by the bandwidth demand at the bottleneck. It depends on all connections sharing the bottleneck and quickly varies over time.

Congestion:

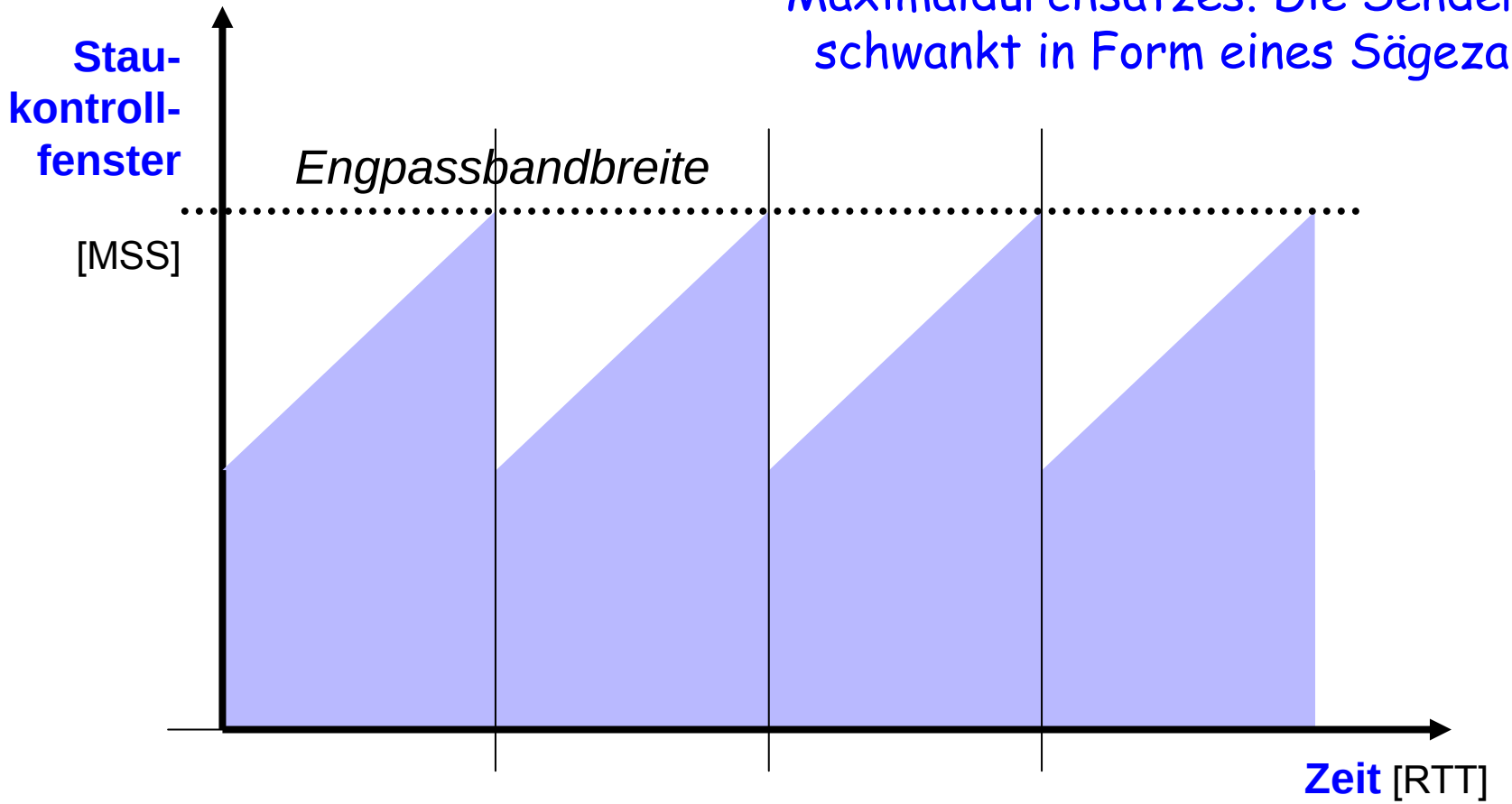
TCP sends more data than the network can handle.



TCP Throughput & TCP Fairness

TCP Durchsatz

TCP* erzielt im Mittel nur 75% des Maximaldurchsatzes. Die Senderate schwankt in Form eines Sägezahns.

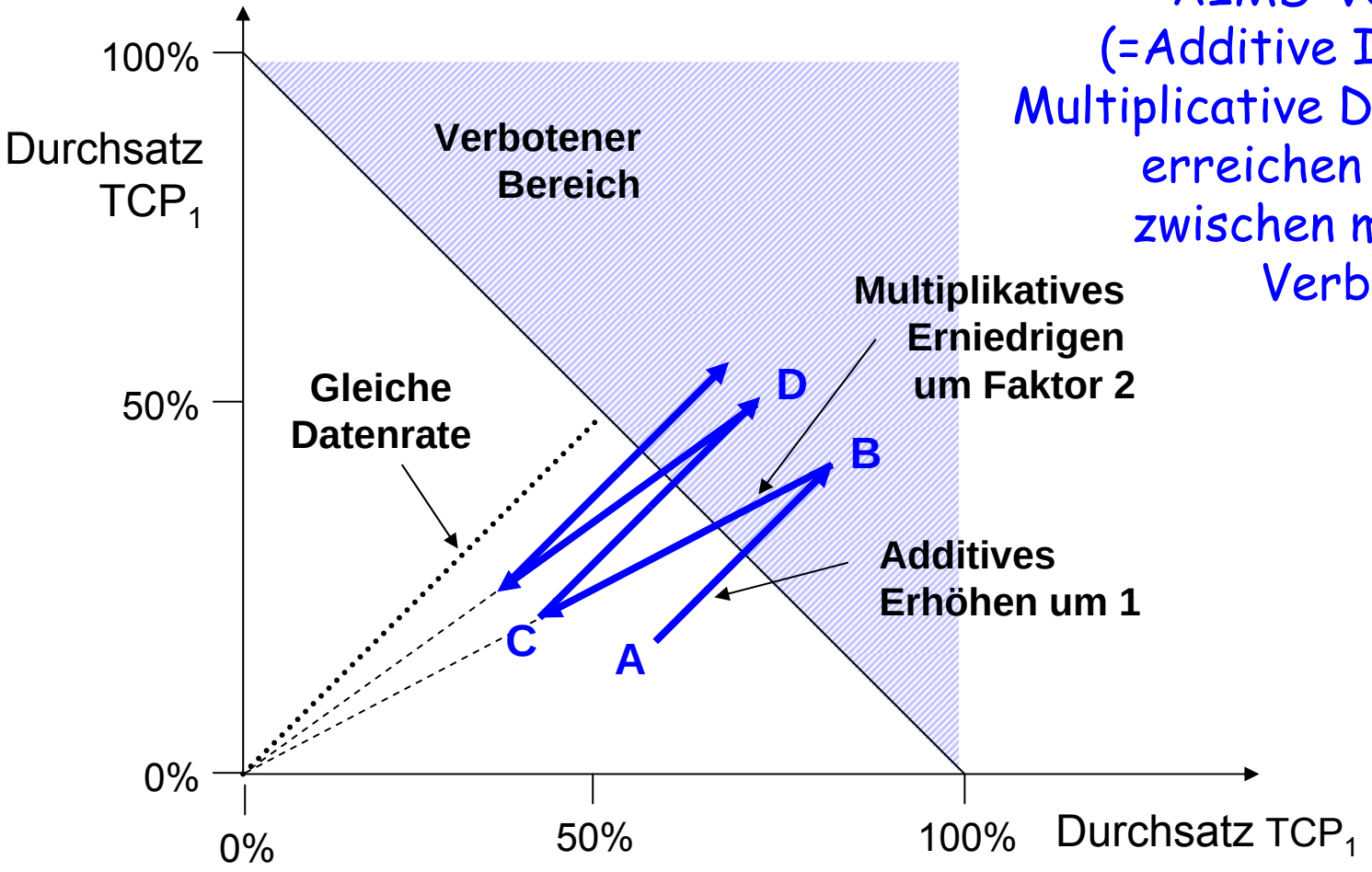


* TCP-Reno, d.h. mit Fast Recovery

TCP Fairness (1)

- Ein Protokoll sollte Fairness gewährleisten. – Aber, was ist Fairness?
  - Keine allgemeine Definition möglich
  - Kontextabhängig
  - Fairness kann (nur) durch gemeinsame Übereinkunft aller an einer Kommunikationssystem beteiligten Partner definiert werden
- Im Internet hat sich der Begriff ***TCP-Fairness*** etabliert:
  - Definiert Senderate in Abhängigkeit der aktuellen Netzwerkbedingungen (Round Trip Time und Paketverlustrate) auf dem jeweiligen Kommunikationspfad
  - Fair ist die Rate, die TCP in Abhängigkeit dieser Bedingungen erreichen würde
- TCP-Fairness beurteilt also Transportprotokolle mit Hilfe der faktisch von TCP erreichten Senderate

TCP Fairness (2)

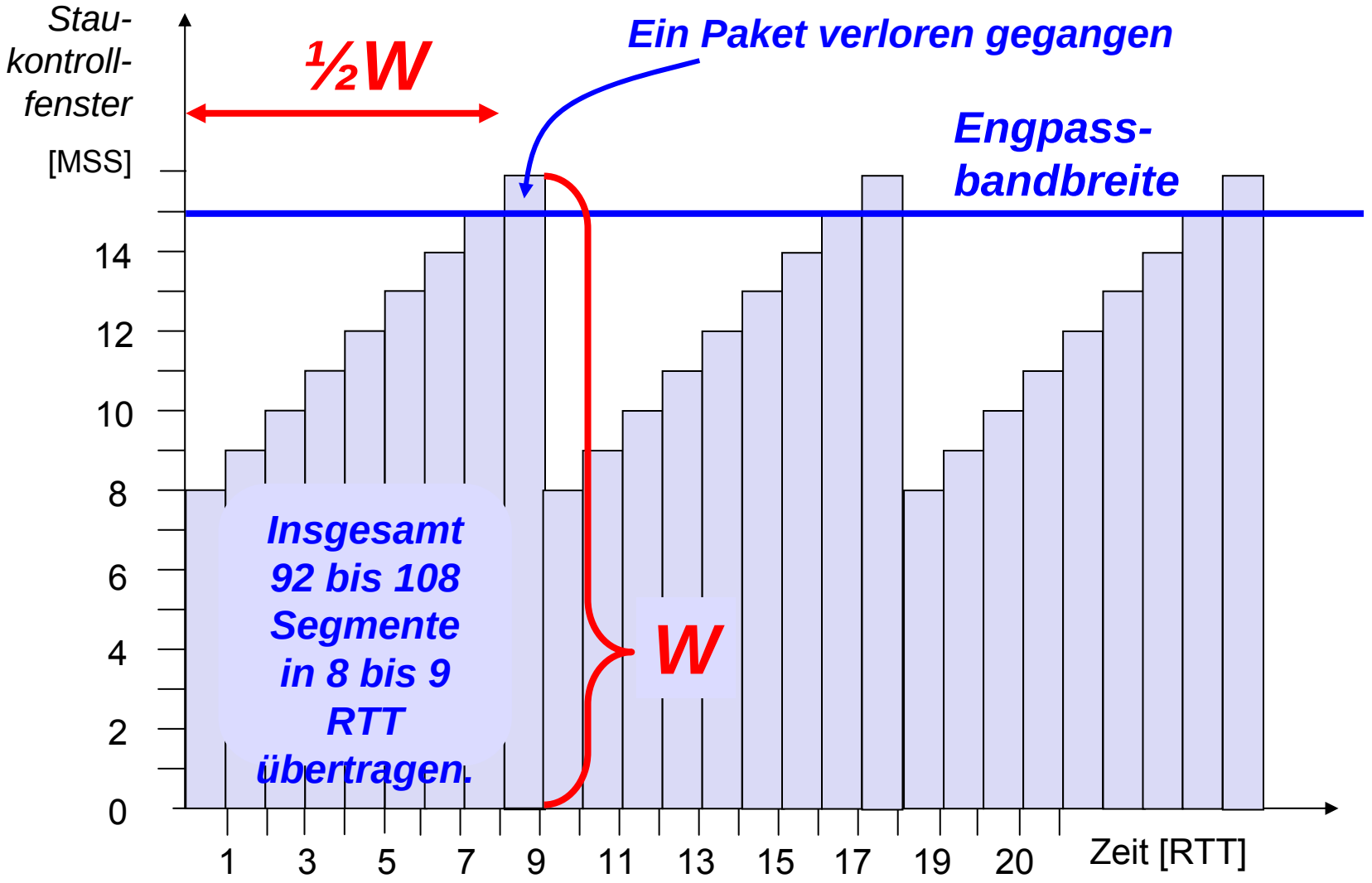


AIMD-Verfahren (=Additive Increase, Multiplicative Decrease) erreichen Fairness zwischen mehreren Verbindungen

TCP Fairness (3)

- TCP Implementierungen sind (im allgemeinen) gleich, d.h. kein Partner wird bevorzugt.
  - TCP teilt die zur Verfügung stehende Bandbreite gleichmäßig auf alle TCP *Verbindungen* auf (vgl. Max-Min-Fair-Share)
  - Achtung: Definitionsgemäße Grundlage von TCP-Fairness ist eine Verbindung, d.h. eine Anwendung mit doppelt so vielen Verbindungen erhält doppelt so viel Bandbreite!
- Probleme entstehen aus der gleichzeitigen Anwesenheit von TCP und anderen Protokollen im Netz
  - Typisches Entwurfsziel: Neue Protokolle sollen TCP-fair sein
  - Dazu erforderlich: Von TCP abstrahierte Beschreibung der von TCP erreichten Senderate (→***TCP-Formel***)
  - Die TCP-Formel basiert auf der Analyse des ***TCP Staukontrollmechanismus***

Herleitung der TCP-Formel (1)



Herleitung der TCP-Formel (2)

$$\text{TCP-Senderate} = \frac{\frac{3}{8}W^2 \cdot \text{MSS}}{\frac{1}{2}W \cdot \text{RTT}} = \frac{3}{4}W \cdot \frac{\text{MSS}}{\text{RTT}}$$

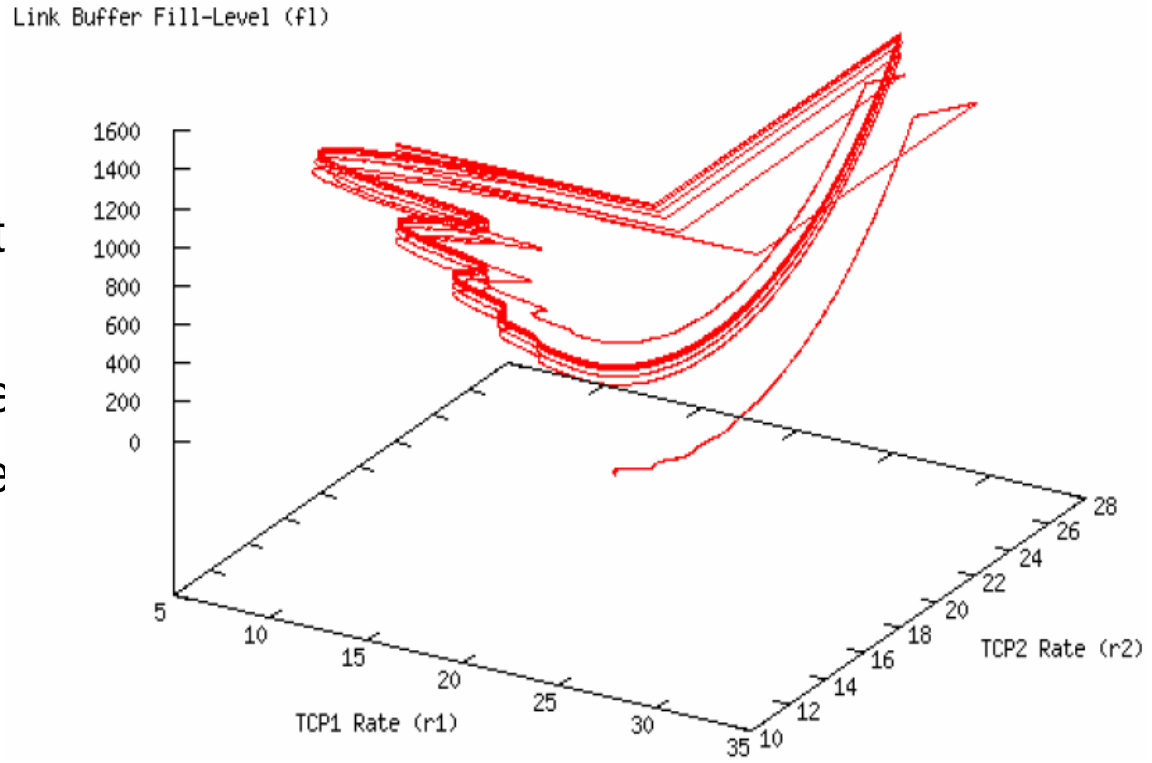
Verlustrate:
(ein Paket von allen gesendeten Paketen)

$$p = \frac{1}{\frac{3}{8}W^2}$$

$$\text{TCP-Senderate} = \sqrt{\frac{3}{2p}} \cdot \frac{\text{MSS}}{\text{RTT}}$$

Some Remarks about TCP Oscillations

- The TCP sending rate oscillates like a jigsaw. Thus TCP uses only 75% of the bottleneck bandwidth on average.
- When only few TCP connections share the bottleneck, the TCPs happen to synchronize.
- With larger multiplexing
- and shorter connections,
- the synchronization effect
- becomes smaller.
- Thus oscillations might be
- a problem in access route
- not in the backbone.



Ratenbasierte Staukontrolle

- Mit der TCP-Formel ist ein alternativer Ansatz möglich, die TCP-Oszillationen zu vermeiden:
  - Messe die Paketverlustrate (p) und Paketumlaufzeit (RTT) und messe bzw. schätze die Segmentgröße (MSS)
  - Die TCP-Formel gibt dann die TCP-faire Senderate.
  - Diese Rate kann mit einem Leaky-Bucket umgesetzt werden.
  - Während der Übertragung werden Verlustrate und RTT gemessen und die Rate allmählich an etwaige Veränderungen angepasst.
- Dieser Ansatz ist besonders für Multimedia-Datenströme geeignet:
  - Daten laufen gleichmäßig statt in Bursts.
- Mit der TCP-Formel kann TCP-faires Mulitcast realisiert werden.

Token Bucket & Leaky Bucket

Token Bucket

- Zähler („Eimer“) mit max. Größe B
- Zähler wird mit Rate R erhöht
- Wenn ein Paket der Größe L gesendet wird, wird der Zähler um L verringert
- Ist der „Eimer leer“, d.h. der Zähler kleiner L, wird das Paket gepuffert, bis der Zähler wieder groß genug ist.
- Garantiert die Einhaltung der mittleren Senderate R

Leaky Bucket

- Puffer („Eimer“) mit max. Größe B
- Pakete werden in den Puffer gepackt und „tröpfeln“ mit der Rate R heraus, d.h. ein Paket der Größe L wird erst gesendet, wenn L Byte aus dem „Eimer getropft“ sind
- Garantiert die Einhaltung der Senderate R und verhindert Bursts

Token Bucket und Leaky Bucket werden oft in Zwischensystemen eingesetzt, um Datenströme an einen „Verkehrsvertrag“ anzupassen.

Fragen

- Erläutern Sie wie TCP einen Staukollaps im Internet auslösen kann!
- Was leistet das Self-Clocking? Warum kann es dennoch keinen Stau verhindern?
- Welche Änderungen bringt TCP Tahoe zur Stauvermeidung?
- Welchen Vorteil bringt Reno gegenüber Tahoe?
- Erläutern Sie, warum sich zwei TCP-Verbindungen die Bandbreite fair teilen können!
- Leiten Sie die TCP Formel her!
- Erläutern Sie die Funktionsweise von ratenbasierter Staukontrolle!

TCP Support in the Network Layer

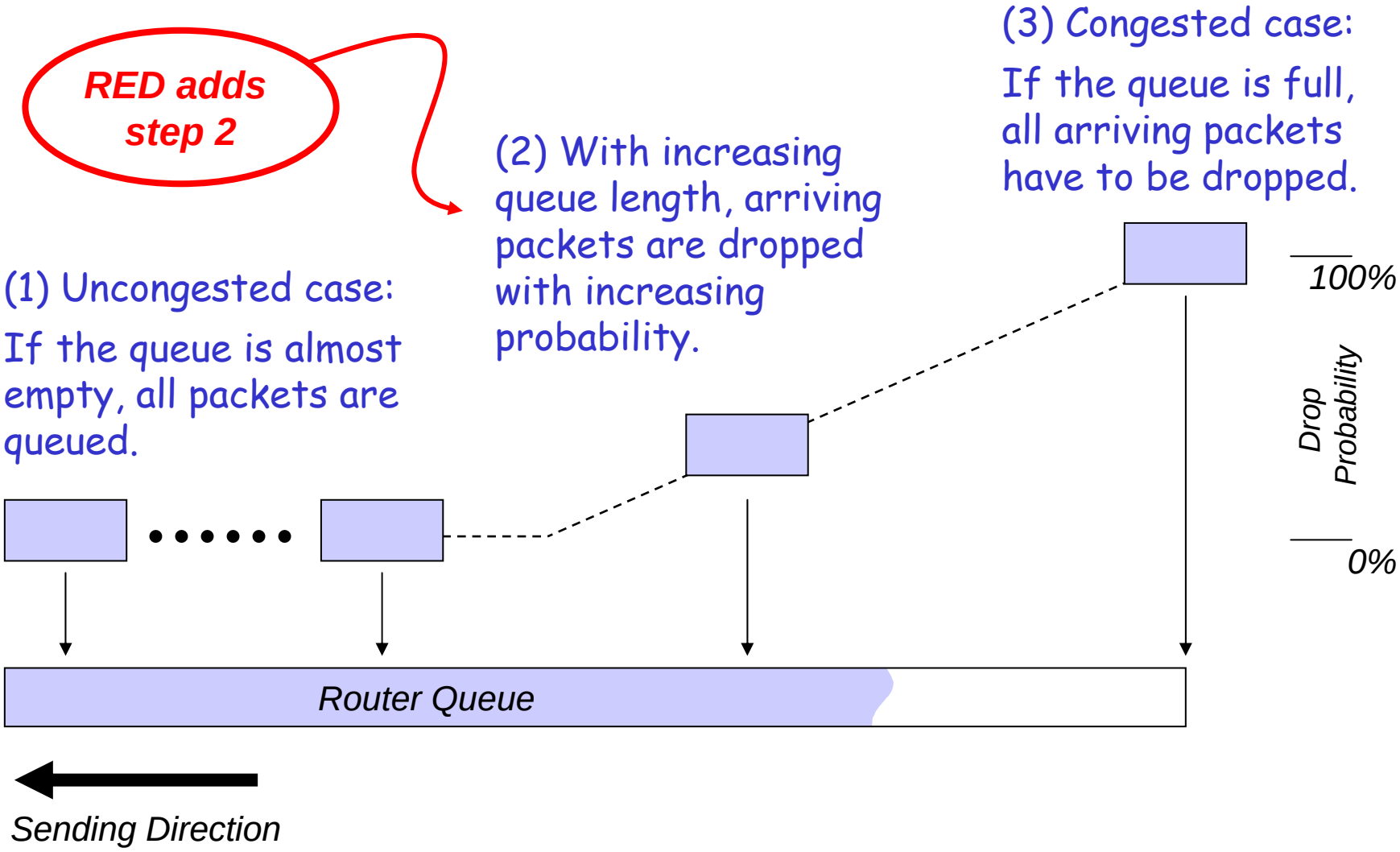
TCP Unterstützung im Netz (1)

Genaue Untersuchungen zeigen, dass es vorteilhaft ist, die Puffer der Router nicht immer ganz auszuschöpfen, bevor die TCP-Senderate reduziert wird.

- Random Early Detection (RFC 2309) ist ein Verfahren, das aktives Warteschlangenmanagement in den Routern anwendet
- Mit steigender Länge der Warteschlange werden Pakete mit steigender Wahrscheinlichkeit verworfen
- Dadurch reduzieren zufällig ausgewählte einzelne TCP-Verbindungen ihre Senderate, anstatt dass alle Verbindungen gleichzeitig ihre Rate reduzieren

Ursprünglich wurde das Verfahren als „Random Early Drop“ (RED) bezeichnet. In Verbindung mit „Early Congenstion Notification“ (ECN) müssen Paket aber nicht verworfen werden (siehe unten). Man hat den Namen daher so geändert, dass die Abkürzung RED erhalten blieb, aber das Wort „drop“ ersetzt wurde.

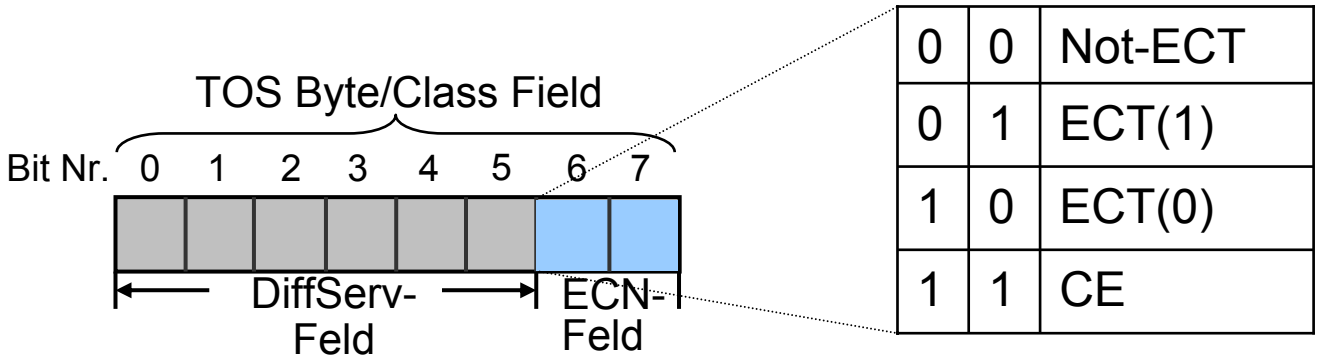
Random Early Drop



TCP Unterstützung im Netz (2)

- Ohne ECN ist das Netzwerk für TCP eine „Black Box“. Die Endsysteme können nur indirekt über Paketverlust auf Stausituationen schließen.
- Explicit Congestion Notification (RFC 3168, Sep. 2001)
  - Vermeidet Paketverluste durch explizite Stauanzeige des Netzes: Statt ein Paket frühzeitig zu verwerfen wird das ECN-Bit im Header gesetzt.

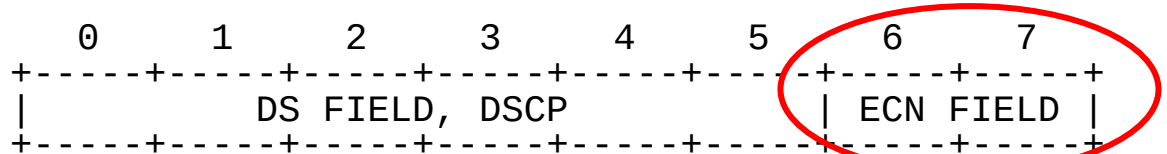
  TCP reagiert darauf mit Reduktion seiner Senderate.
  - ECN setzt Active Queue Management (z.B. RED) im Router voraus.
  - Markierung des IP-Pakets mittels Congestion Experienced (CE) Bit.
  - Diese Anzeige muss erfolgen, bevor Warteschlange wirklich voll ist.
  - ECN-Fähigkeit muss signalisiert werden, um TCP Fairness sicherzustellen: ECN-Capable Transport (ECT) Bits



0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Explicit Congestion Notification (1)

IP Header

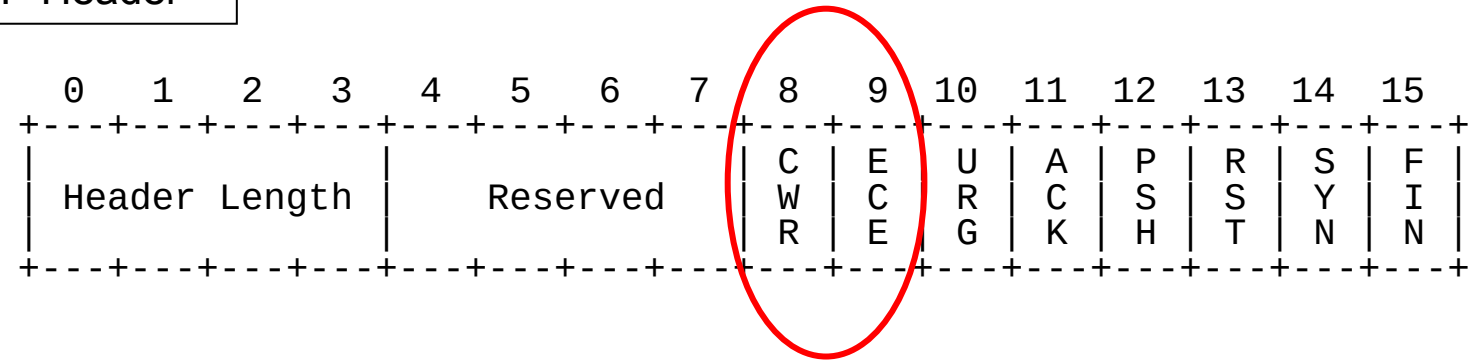


DSCP: differentiated services codepoint

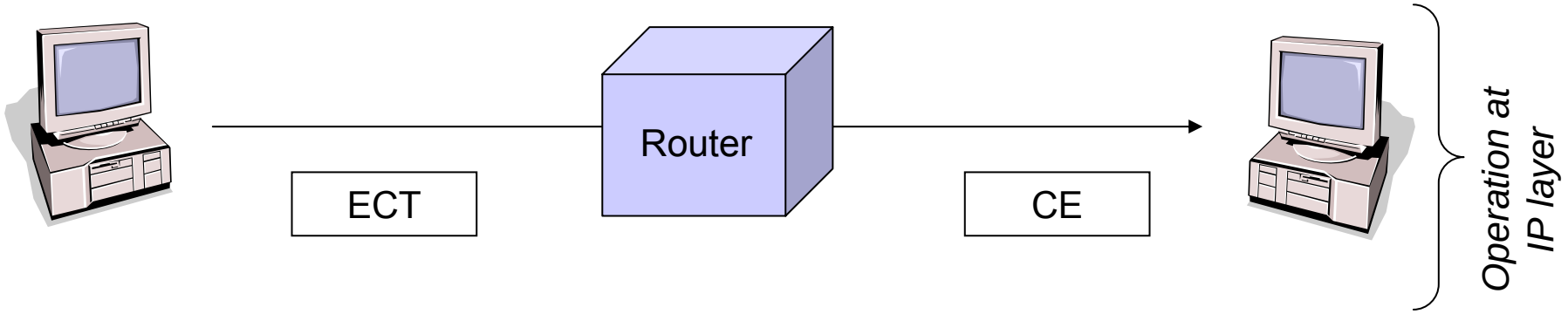
ECN: Explicit Congestion Notification

0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

TCP Header

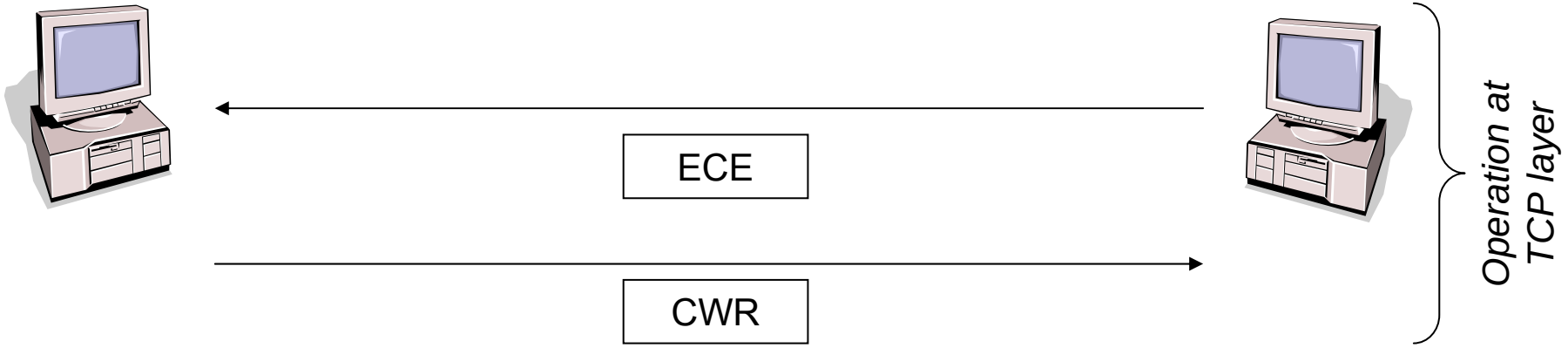


Explicit Congestion Notification (2)



1. ECN capable hosts set ECT in the IP header. ECN capable routers set ECT to CE to indicate congestion.

2. ECN capable receivers echo the CE in the TCP header as ECE (=echo congestion experienced). ECN capable senders acknowledge the reception ECE as CWR (=congestion window reduced).



More TCP Design Issues …

(How to cheat with TCP)

Misbehaving TCP Empfänger (1)

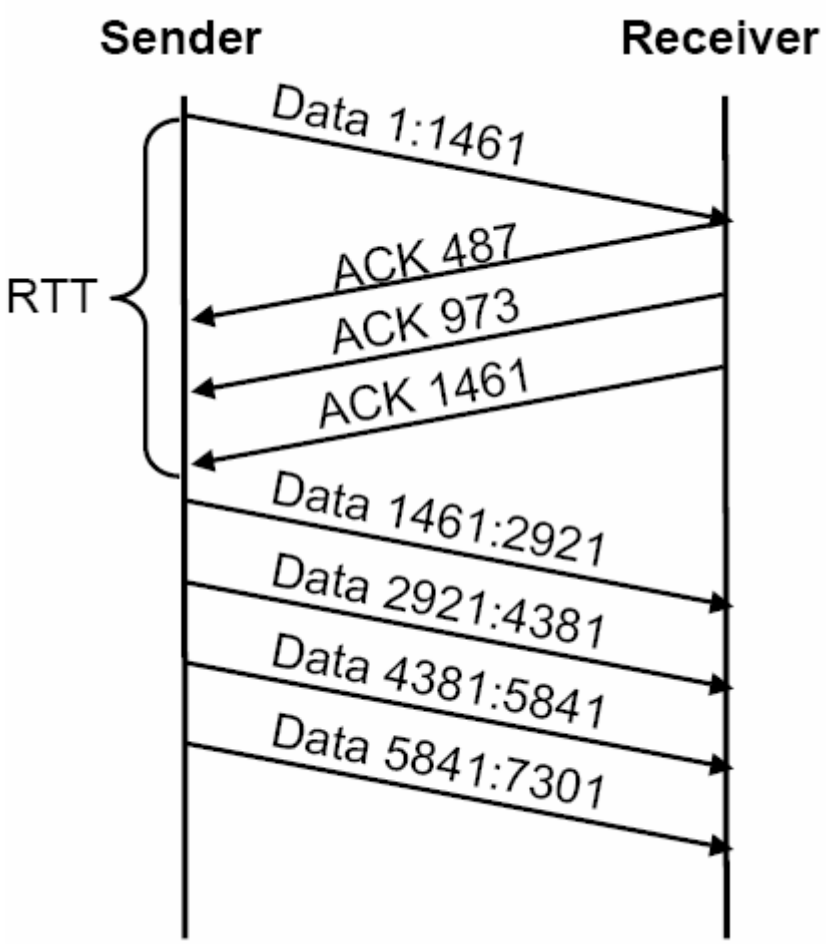
- TCP ist anfällig gegen unfairen Bandbreitenkonsum:
  - Die Möglichkeit zu unfairem Verhalten ist Best Effort Netzen inhärent, aber TCP ermöglicht es auch einer einzelnen Partei (und zwar dem Empfänger), sich einen Vorteil zu verschaffen.
  - Ein entsprechender Web-Browser kann also z.B. schneller Web-Seiten saugen, ohne dass er dafür einen Web-Server ändern müsste.
  - Aus der Spieltheorie ist bekannt, dass sich dieses Verhalten durchsetzt, da es ja nicht von der Gruppe kontrolliert werden kann.
- Der Ansatzpunkt für diesen „Betrug“ sind Inkonsistenzen in der TCP-Spezifikation.

Misbehaving TCP Empfänger (2)

- TCP Sequenznummern zählen Bytes, das Staukontrollfenster zählt in Segmenten.
- Angriff:

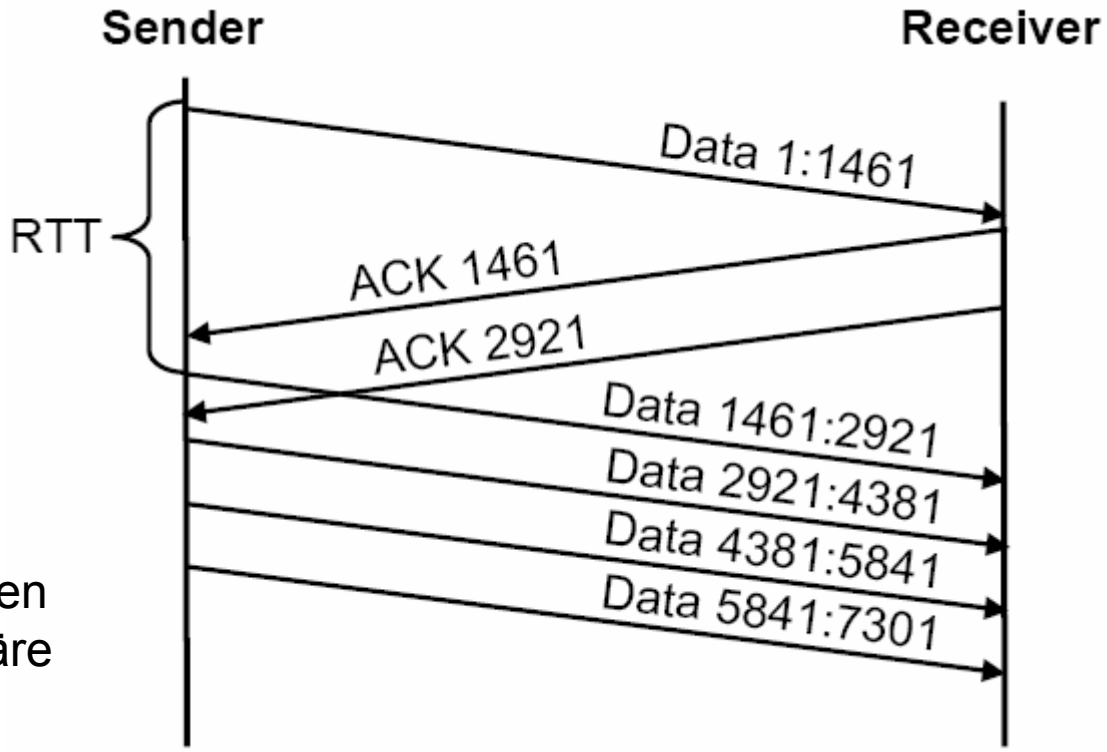
  Bestätige jedes Segment in einzelnen Häppchen!
- Dadurch öffnet sich das Staukontrollfenster schneller.
- Bemerkung:

  Würde das Staukontrollfenster ebenfalls in Bytes zählen, wäre der Angriff unmöglich.



Misbehaving TCP Empfänger (3)

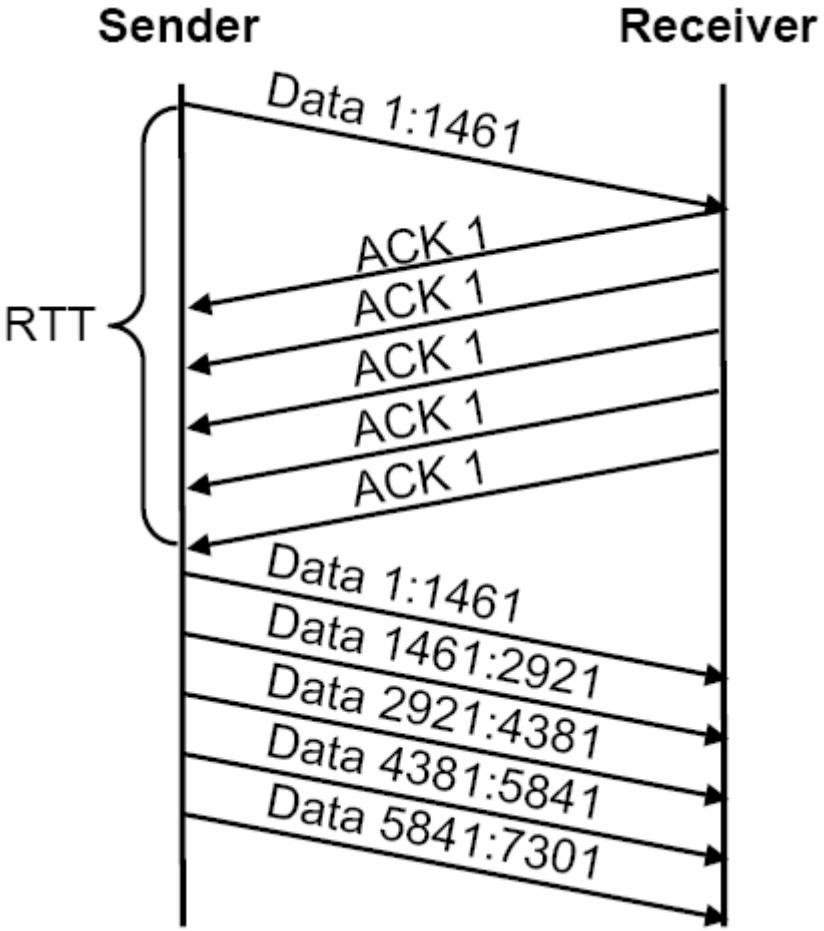
- Paketverluste sind selten und Quittungen beweisen nichts.
- Angriff:
  - Bestätige Segmente schneller als sie eintreffen.
- Bemerkung:
  - Würden Quittungen tatsächlich den korrekten Empfang beweisen, wäre der Angriff unmöglich.



Anders als im ersten Fall wäre zur Lösung hierfür eine Protokolländerung erforderlich. Alternativ würde aber bereits eine Erhöhung der Paketverlustwahrscheinlichkeit, d.h. das gelegentliche auslassen eines Segments beim Sender, einen solchen Angriff unrentabel machen.

Misbehaving TCP Empfänger (4)

- Duplicate ACKs vermischen negative und positive Quittungen:
  - Segment x soll wiederholt werden
  - Segment x+1, x+2, x+3, … wurde empfangen
- Gemäß TCP Reno läuft das Self-Clocking weiter und auch das Staukontrollfenster wird weiter geöffnet, d.h. es werden immer weiter neue Segmente übertragen bis das Flusskontrollfenster erschöpft ist.



TCP Sockets

Geschichte von BSD UNIX und TCP/IP (1)

April 1981 – DARPA und die Universität Berkeley beginnen ein gemeinsames Projekt, um BSD mit besseren Dateisystemen, Interprozesskommunikation und Netzwerkfunktionalität auszustatten.

April 1982 – Die Version 4.1a BSD enthält nun auch TCP/IP, d.h. Anschluss ans ARPAnet. Tests der Implementierung finden in Berkely selbst statt.

August 1983 – Release von 4.2 BSD, über 1000 Lizenzen verkauft. AT&T's System V übernimmt die Netzwerkfunktionalität aus BSD. Die Socket-Schnittstelle ist erfolgreich, aber die eigentliche Protokollimplementierung ist veraltet und nicht auf dem Stand der Entwicklung bei BBN.

Juni 1986 – Die Version 4.3 BSD übernimmt die Performanceverbesserungen aus der BBN Implementierung. Tests ergeben, dass BSD stabiler läuft als BBN, aber schlechter mit Stausituationen umgehen kann.

Oktober 1986 – Der ***große Stau-Kollaps im Internet***: Durchsatz zwischen Lawrence Berkeley Laboratory und der Uni Berkeley (Entfernung 400 yards, zwei IMP Hops) fällt um drei Größenordnungen auf nur noch 40 Bit/s.

Quellen: K. McKusik „20 Years of Berkeley“ in „OpenSources“ O'Reilly
und W. R. Stevens, „TCP/IP Illustrated – vol 2“, Addisson Wesley

Geschichte von BSD UNIX und TCP/IP (2)

Juni 1988 – Portierung von VAX auf andere Architektur (4.3 BSD Tahoe). Die TCP-Implementierung integriert jetzt: Slow-start, Congenstion Avoidance und Fast Retransmit.

Januar 1990 – Die neue Release 4.3 BSD Reno integriert auch Verbesserungen am Netzwerkprotokollstapel: Fast Recovery, TCP Header Prediction, SLIP header compression, …

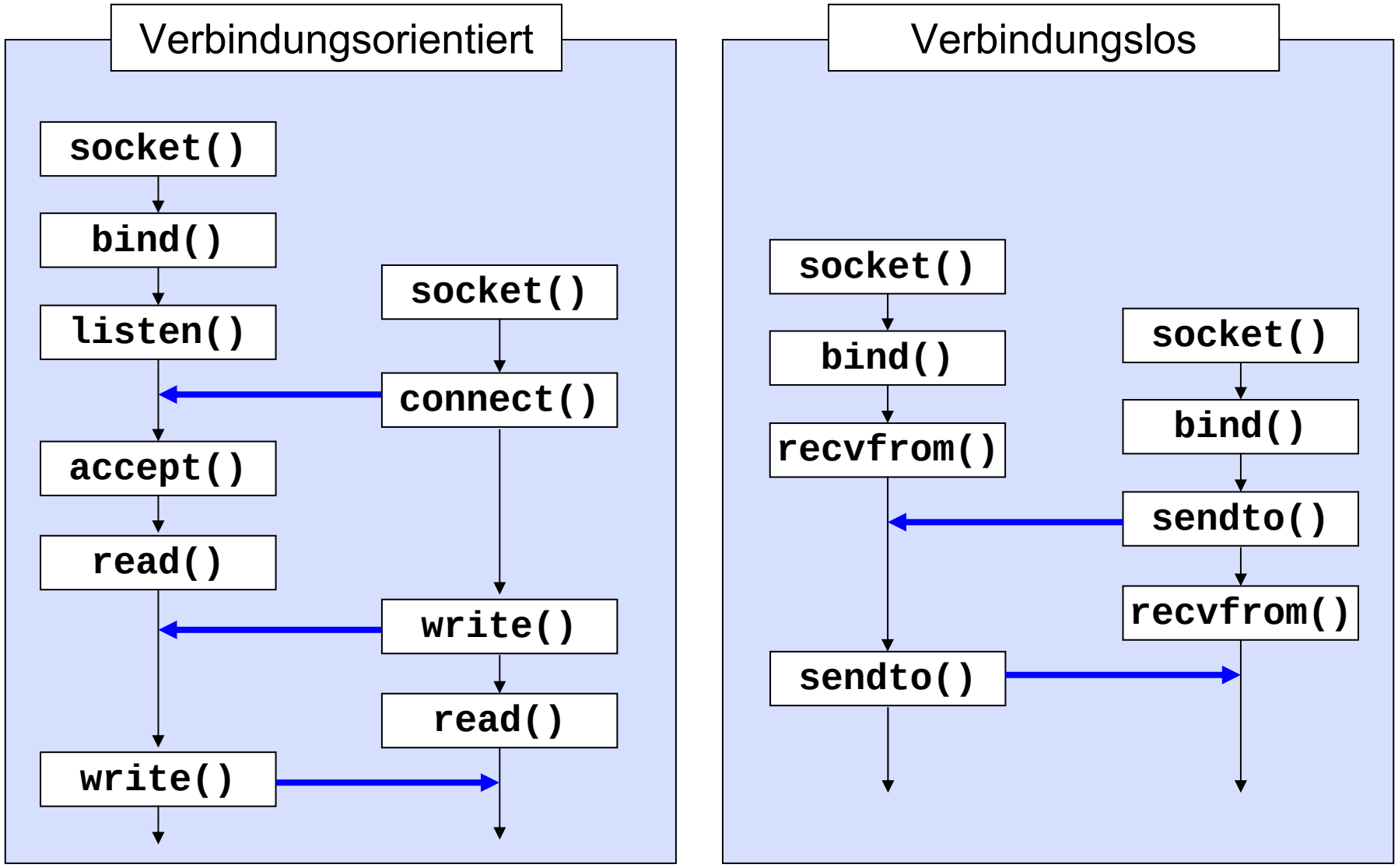
Juni 1993 – Die Version 4.4 BSD erweitert den TCP/IP Stack u.a. um Multicast.

Oktober 1994 – TCP Vegas verwendet feinere Timer, um ggf. schon beim ersten DupACK eine Übertragungswiederholgung anzustoßen und bei Erhöhung der RTT die Senderate zu drosseln, d.h. proaktive Staukontrolle statt nur reaktive Staukontrolle.

Oktober 1996 – Selective Acknowledgement (RFC 2018) nutzt eine neue TCP Option, um den Verlust einzelner Segmente anzuzeigen.

April 1999 – TCP New Reno (RFC 2581) beseitigt Unklarheiten aus Reno (RFC 2001) und spezifiert damals offen gelassene Punkte, z.B. Verhalten nach langen Phasen der Inaktivität einer TCP-Verbindung.

Socket Schnittstelle – Überblick (1)



Socket Schnittstelle – Überblick (2)

Verbindungsorientiert:

- Passive Seite: Server bindet sich mittels *bind()* an einen bestimmten Port und erwartet eingehende Verbindungen mittels *listen()*.

  Solche Server Ports (engl. „well-known ports“) sind meist einer bestimmten Anwendung zugeordnet und können oft nur von Anwendungen mit Root-Rechten geöffnet werden. Diese Konvention ist aber nicht durch TCP vorgegeben. TCP unterscheidet nur das aktive und das passive Öffnen.

- Aktive Seite: Client erstellt mittels *connect()* eine Verbindung zum Port des Servers. Über diese Verbindung kann im Anschluss kommuniziert werden.

  Solche Client Ports werden meist nur kurze Zeit offen gehalten (engl. „ephemeral ports“). Die Portnummern dafür werden aus einem Bereich gewählt der allen Anwendungen zugänglich ist. Typischerweise überlässt die Anwendung dem Betriebssystem die Wahl eines Ports. Durch bind() oder connect() wird die Socket der entsprechenden Anwendung zugeordnet.

Verbindungslos:

- Beide Seiten binden sich an einen bestimmten Port und erwarten eingehende Datagramme bzw. senden selbst Datagramme.

Socket – UNIX Manual Excerpt (1)

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Socket creates an endpoint for communication and returns a descriptor. The ***domain*** parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in **<sys/socket.h>**. The currently understood formats include:

```
PF_UNIX,PF_LOCAL   Local communication
PF_INET            IPv4 Internet protocols
PF_INET6           IPv6 Internet protocols
PF_IPX             IPX - Novell protocols
PF_NETLINK         Kernel user interface device
PF_PACKET          Low level packet interface
```

Socket – UNIX Manual Excerpt (2)

The socket has the indicated ***type***, which specifies the communication semantics. Currently defined types include:

SOCK_STREAM
Provides sequenced, reliable, two-way, connection- based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK_DGRAM
Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_RAW
Provides raw network protocol access.

SOCK_SEQPACKET
Provides a sequenced, reliable, two-way connection- based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each read system call.

Some socket types may not be implemented by all protocol families; for example, **SOCK_SEQPACKET** is not implemented for **PF_INET**.

Socket – UNIX Manual Excerpt (3)

The ***protocol*** specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. The protocol may be zero in that case. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner.

RETURN VALUES

-1 is returned if an error occurs; otherwise the return value is a descriptor referencing the socket.

EPROTONOSUPPORT

The protocol type or the specified protocol is not supported within this domain.

ENFILE

Not enough kernel memory to allocate a new socket structure.

…

The Socket Interface is very closely related to UNIX and C. Socket support in other operating systems and programming languages reflects this historical development.

Bind – UNIX Manual Excerpt (1)

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sid,
         const struct sockaddr *ptr,
         socklen_t len);
```

Bind() assigns a name to an unnamed socket so that other processes can connect to the socket. In the Internet domain this name is an address & port pair and processes are typically remote processes.

sid is the identifier returned by the socket() call.

ptr points to address family dependent structure.

len determines the size of this structure.

Bind – UNIX Manual Excerpt (2)

Internet Sockets

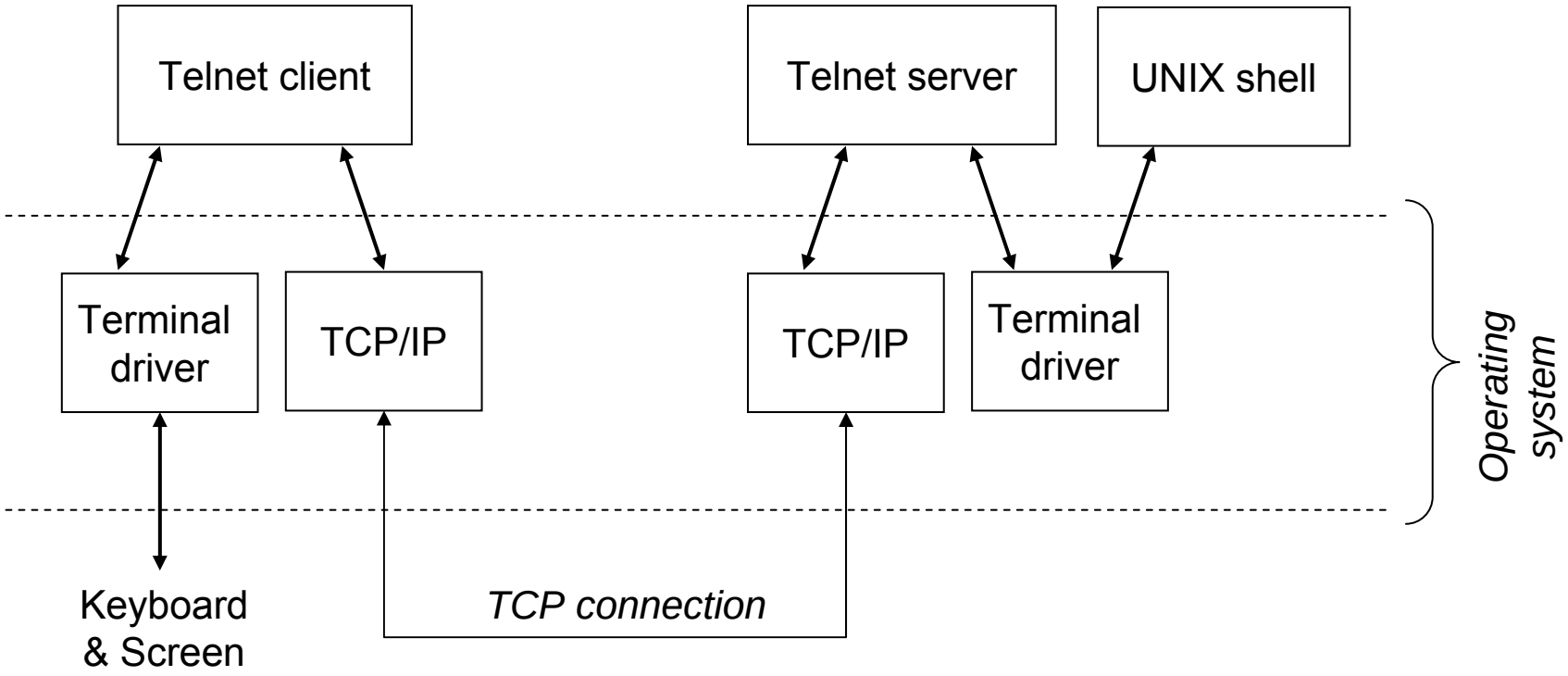
```
struct sockaddr_in {
  sa_familiy_t sin_familiy;  /* = AF_INET */
  in_port_t    sin_port;     /* 16-bit port number */
                             /* network byte ordered */
  struct in_addr sin_addr;   /* 32-bit IPv4 address */
                             /* network byte ordered */
  char         sin_zero[8];
};
```

Unix Sockets

```
struct sockaddr_un {
  uint8_t sun_length;
  short sun_family;   /* = AF_LOCAL */
  char sun_path[100]; /* null terminated path name */
};
```

Simple TCP Applications

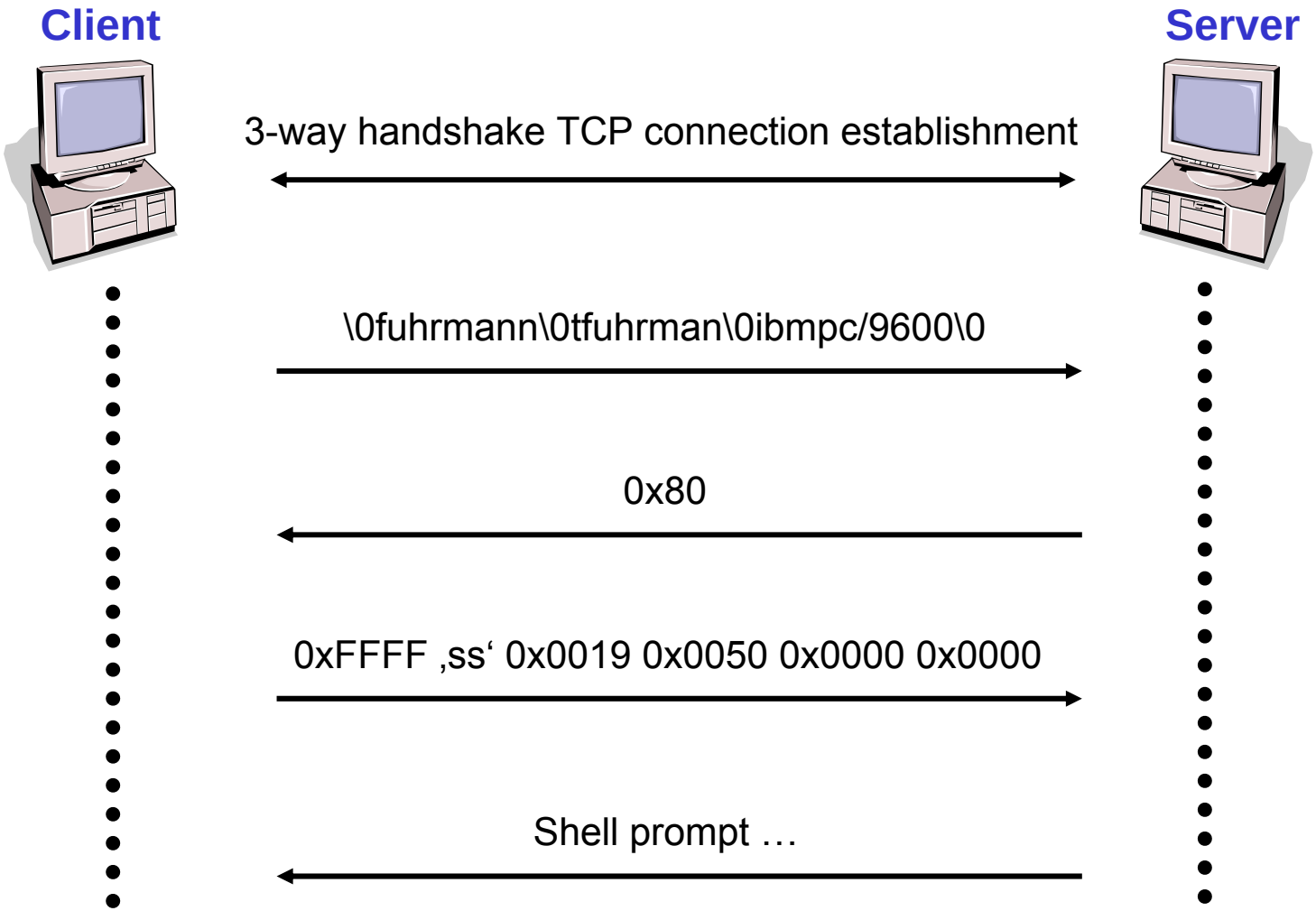
Rlogin & Telnet (1)



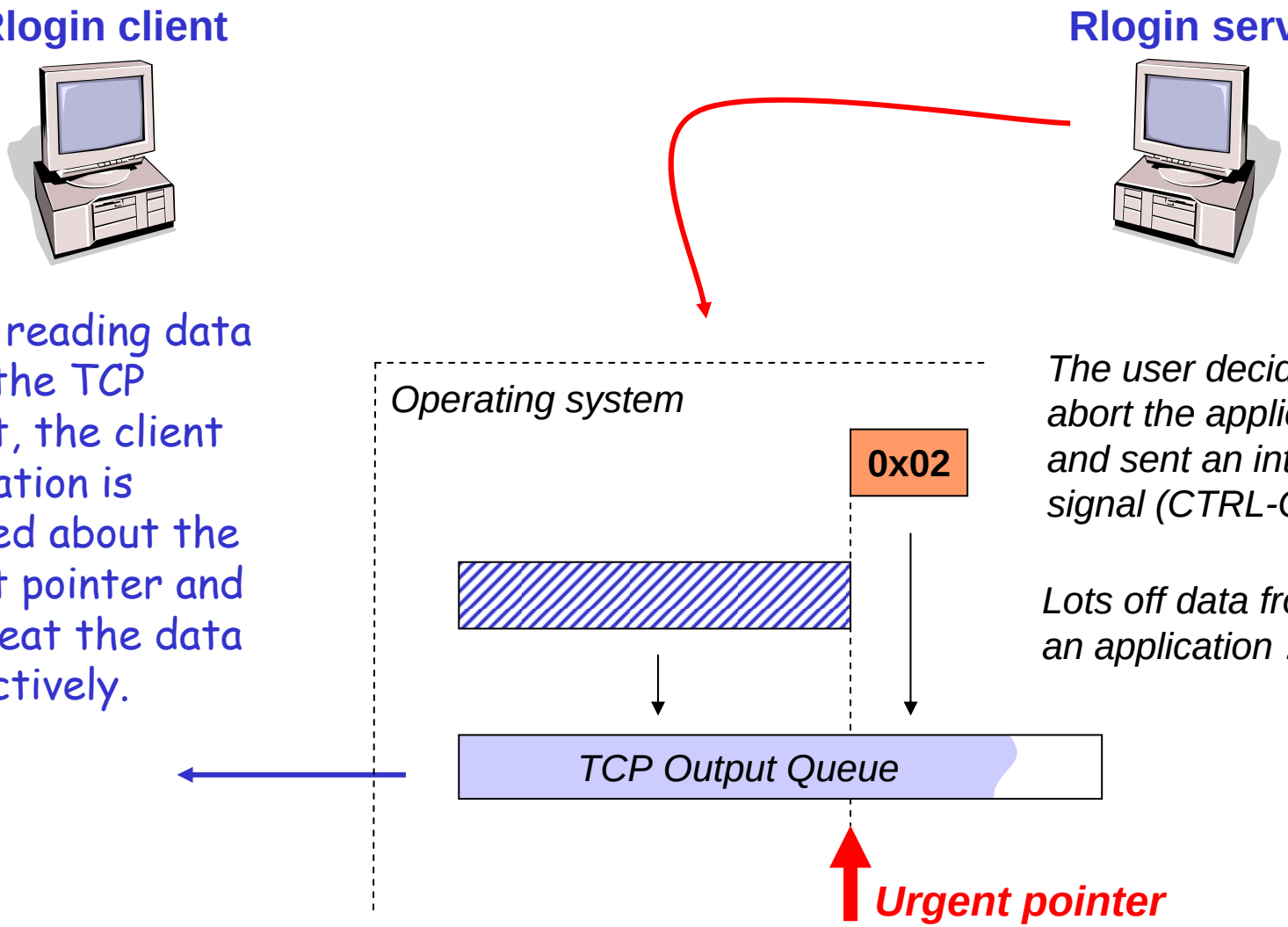
Rlogin & Telnet (2)

- Rlogin & Telnet are simple, useful, but insecure applications.
- Basically, they sit between a terminal driver and a UNIX login shell so that a user can access a server from a remote terminal.
- Rlogin was designed to operate between UNIX hosts only. Telnet provides elaborate mappings to operate between arbitrary machines and terminals.
- Rlogin & Telnet use in-band-signaling, i.e. some ASCII characters are interpreted by the client and server and not forwarded to the login shell.
  - Rlogin example: 0xFF 0xFF <window size>
    The rlogin client informs the rlogin server about the screen size, i.e. how much lines of text with how many characters per line the client can display.
- Rlogin also use the TCP urgent pointer. This is sometimes but incorrectly called out-of-band-signaling. – Examples:
  - 0x80 The rlogin server asks the rlogin client to sent its window size.
  - 0x02 The rlogin client shall discard all data up to this command.
  - 0x20 The rlogin client processes the CTRL-S and CTRL-Q commands itself.

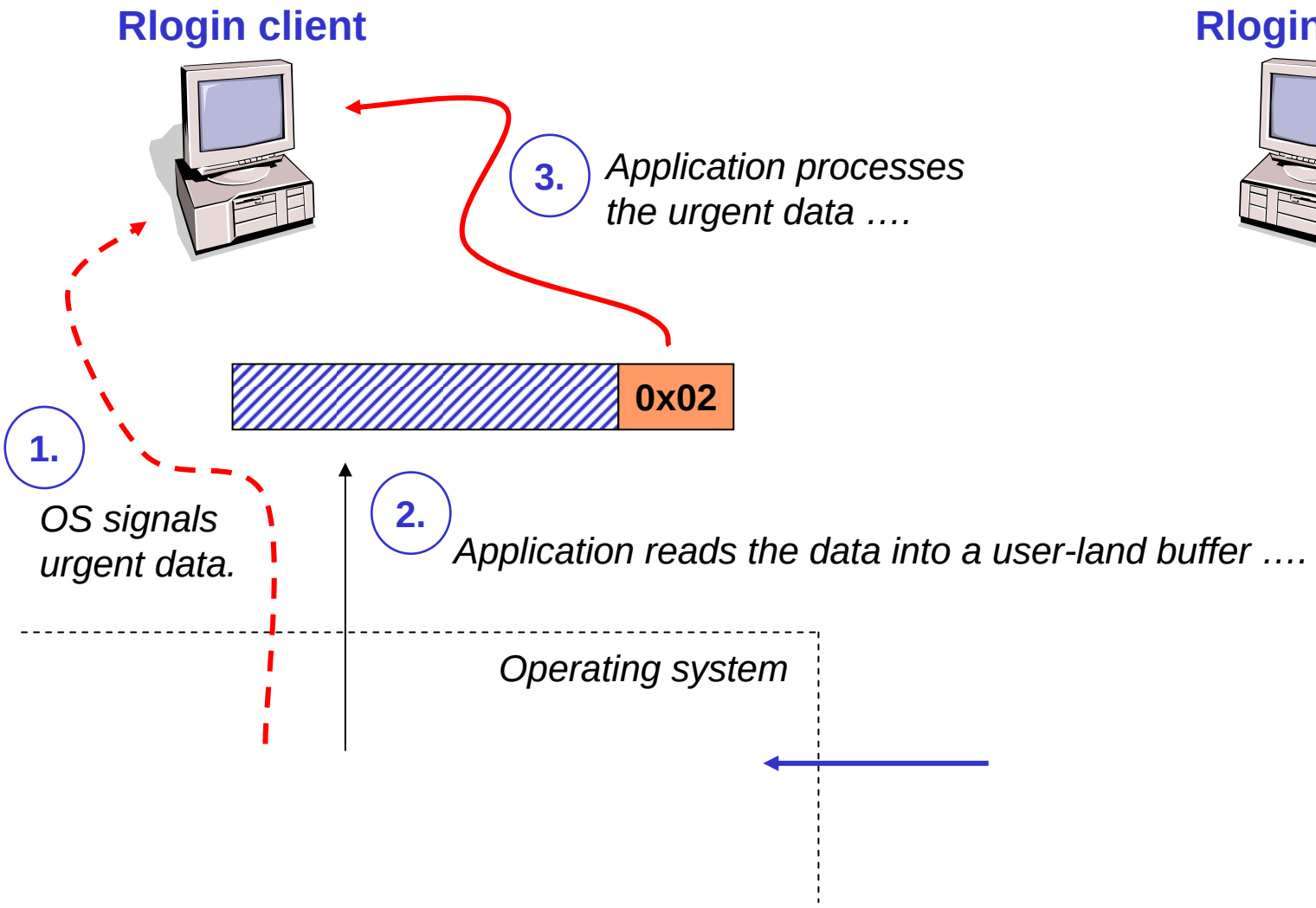
Rlogin Example



Using the Urgent Pointer (1)



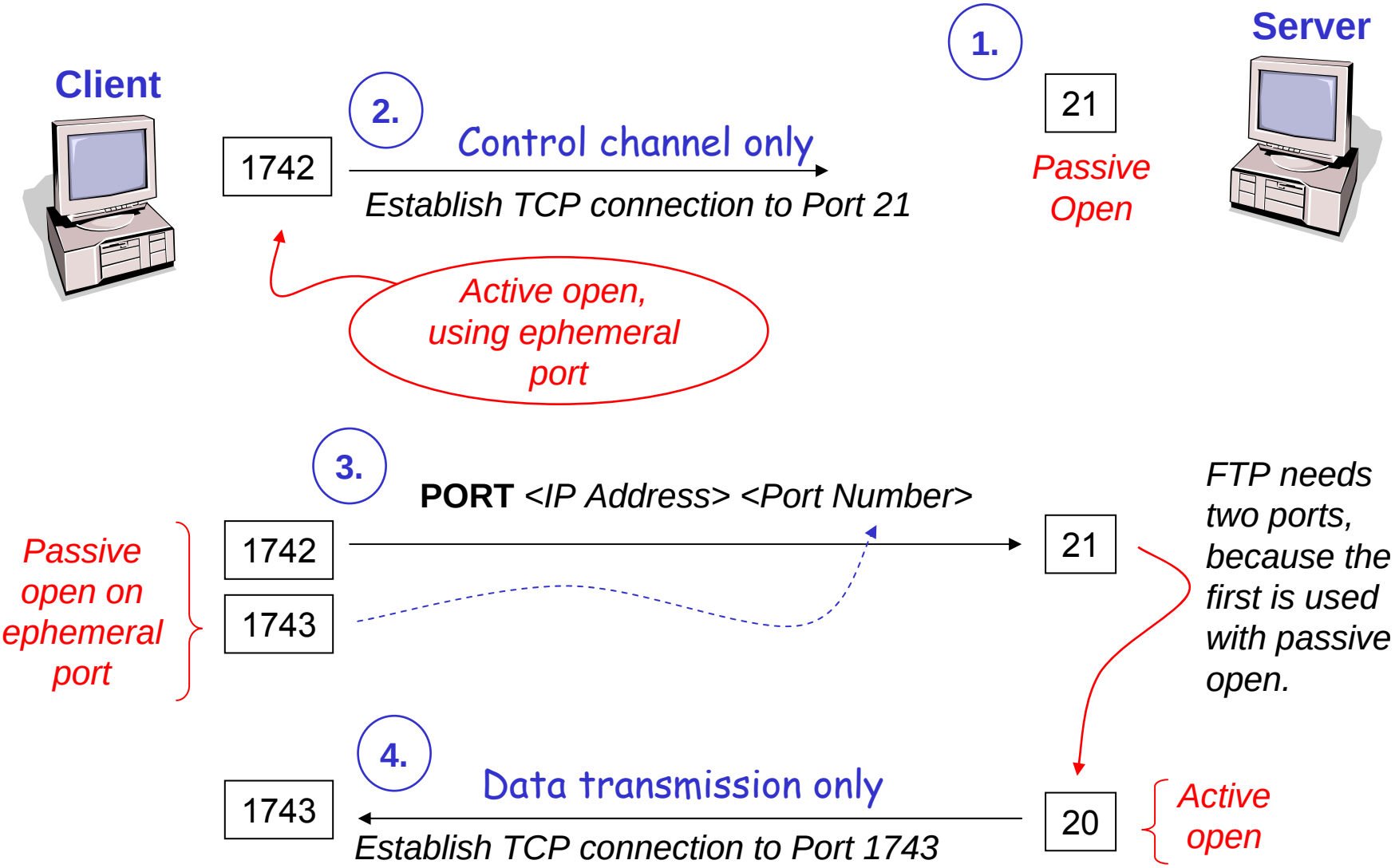
Using the Urgent Pointer (2)



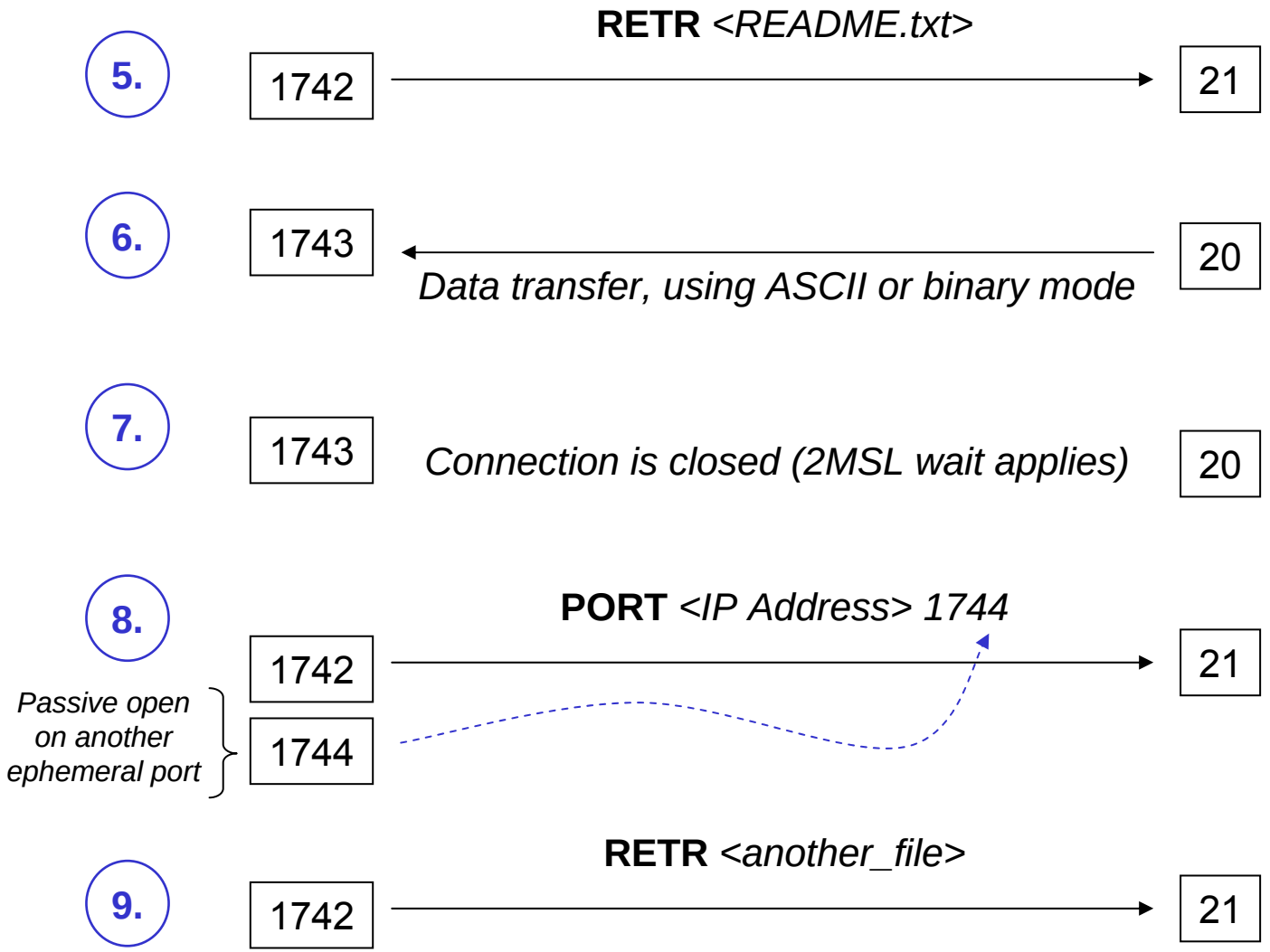
Telnet Example

- Telnet uses in-band-signaling:
  - 0xFF is used as „interpret as command“ (IAC) indication.
  - Since telnet defines a generic „network virtual terminal“ which uses 7-bit ASCII only, this is no problem.
  - But since there is an 8-bit option, 0xFF must be stuffed when this option is in use.
- Examples for telnet commands:
  - EOF (end of file)
  - SUSP (suspend current process)
  - IP (interrupt process)
- Telnet uses four commands (WILL, WONT, DO, DONT) to negotiate options. Examples:
  - IAC WILL <terminal type> means that the sender wants to enable the respective terminal type.
  - IAC DO <window size> means that the sender wants the receiver to set its display parameters (columns, rows) to the given parameters.

File Transfer Protocol (1)



File Transfer Protocol (2)



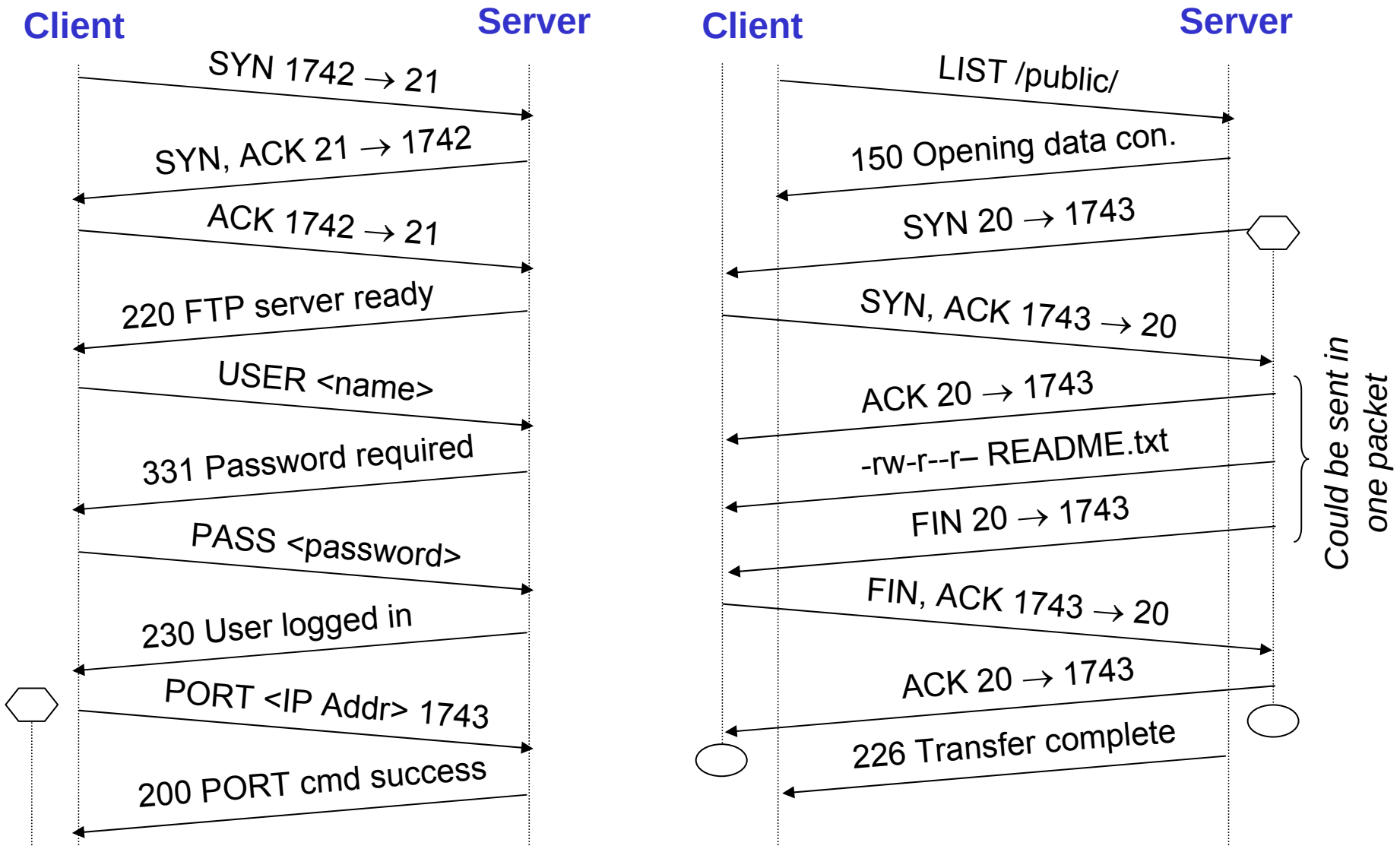
File Transfer Protocol (3)

- FTP server passively opens well-known port and waits for incoming client requests.
- FTP client connects to server and sends commands (see below).
- For data transfer,
  - the client passively opens another port, and
  - sends its number to the server.
  - Then the server connects to this port and
  - Transfers the data.
  - Closing the data connections indicates end of file.
- Using separate data connection simplifies transfer:
  - No multiplexing of control and data traffic required.
  - Ongoing transfer does not interfere with further control command, e.g. commencing further transfers or aborting a transfer.
- Using separate ports and different directions for control and data distorts the clear separation of
  - Server = Passive open = Well known port
  - Client = Active open = Ephemeral port

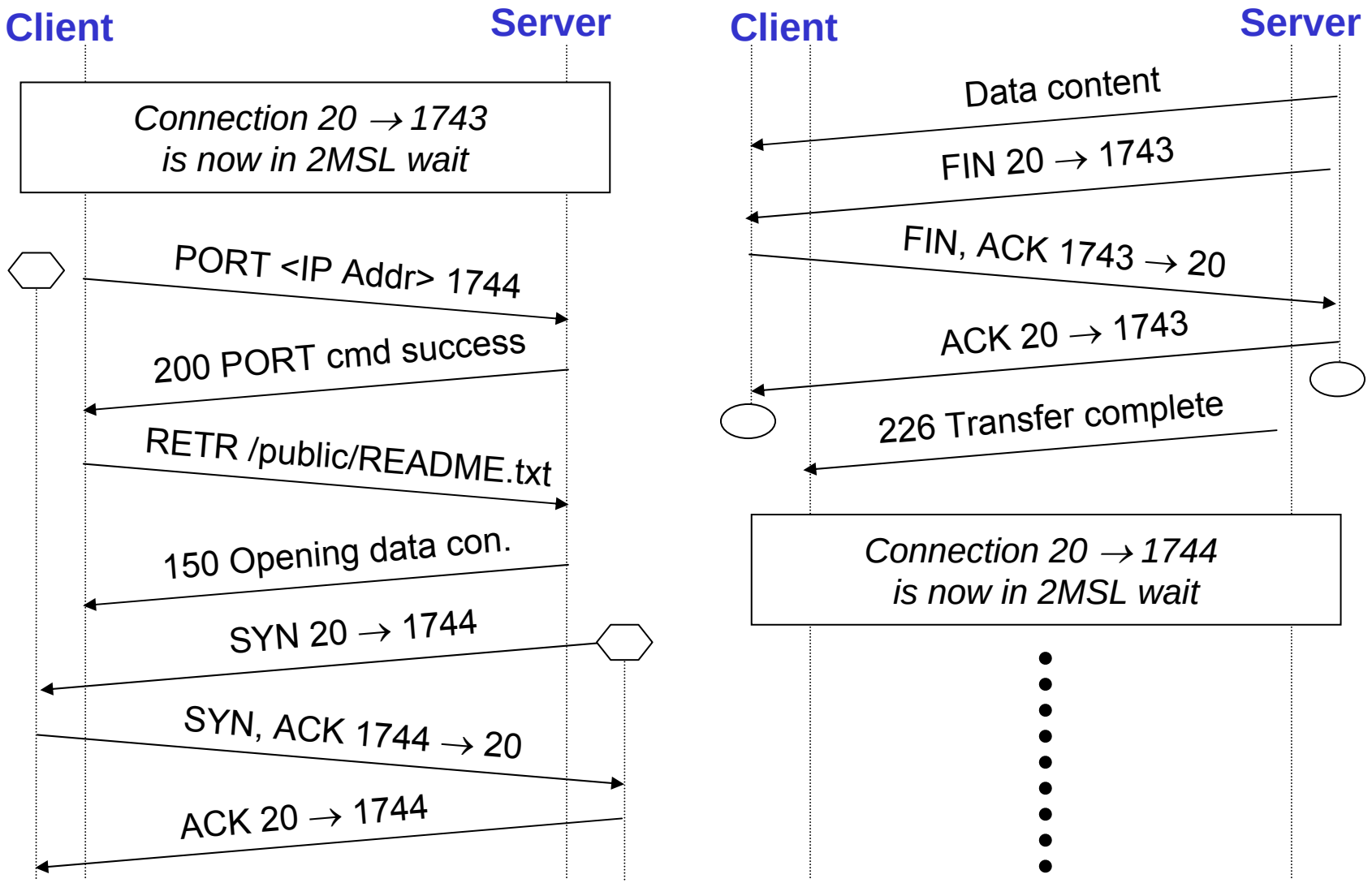
File Transfer Protocol (4)

- Selection of ***FTP Commands*** (sent on control channel)
  - USER <name>
  - PASS <password>
  - LIST <directory>
  - PORT <IP Addr><Port>
  - TYPE <A or I>
  - RETR <filename>
  - QUIT
- Selection of ***FTP Replies*** (received on ctronol channel)
  - 200 Command OK
  - 331 User name OK, password required
  - 425 Cannot open data connection
  - 452 Error writing file
  - 500 Syntax error (unrecognized command)
- These commands (and others not shown here) are sent in plain text ASCII.
- Only the 3-digit decimal reply numbers are processed by an FTP client application.

Watching FTP on Packet Level (1)



Watching FTP on Packet Level (2)



Trivial File Transfer Protocol

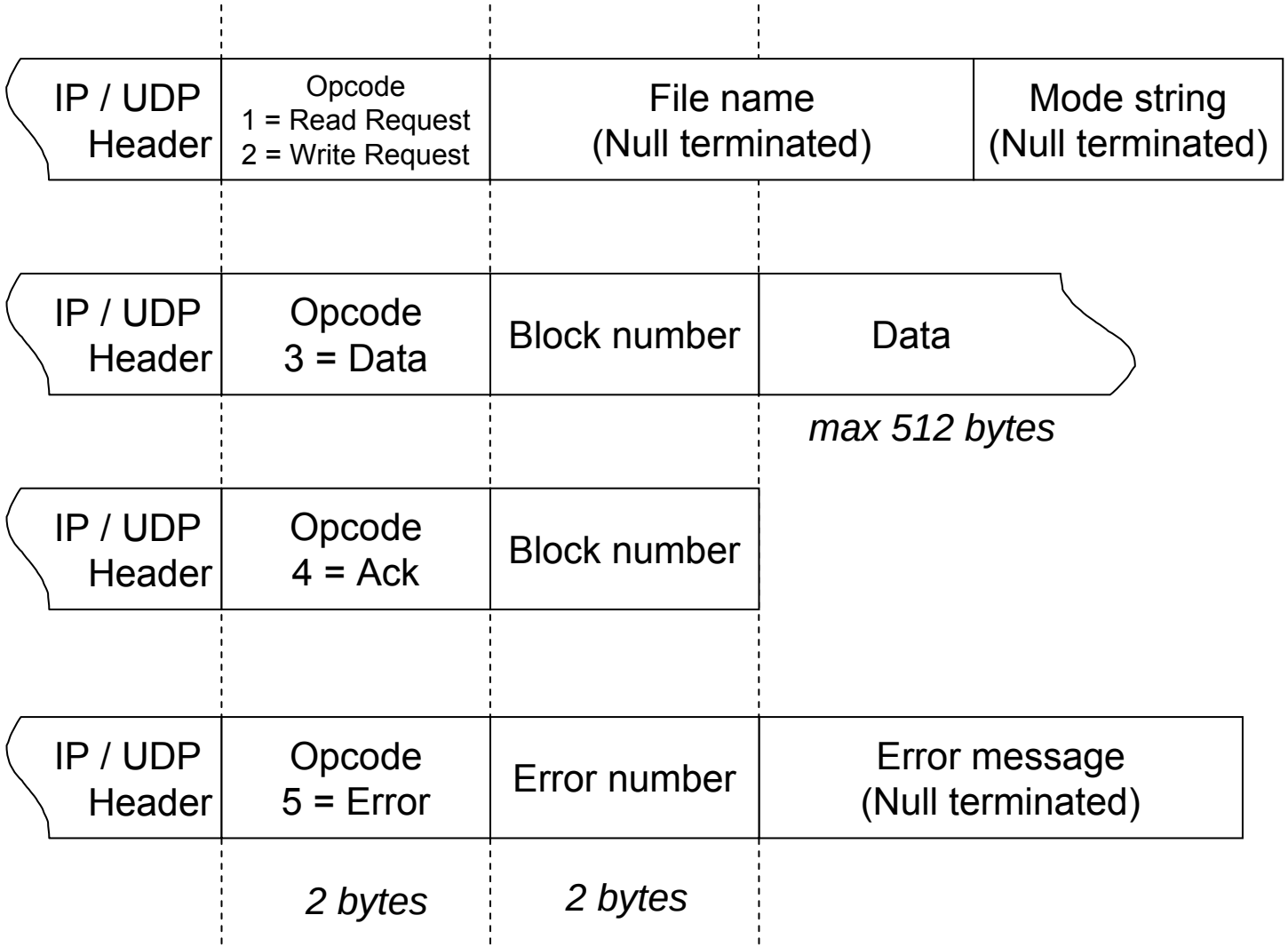
- Very limited functionality, thus simple to implement.
- No authentication mechanism, thus insecure.
- Typically used to fetch a kernel image for booting a diskless computer.

- TFTP server binds to UDP port 69.
- Transfers files of up to 32 MB with a stop & wait protocol.
- Only trivial file transfer that is read or write a file.
- TFTP provides nothing else, not even listing a directory, etc. ($\rightarrow$ FTP)

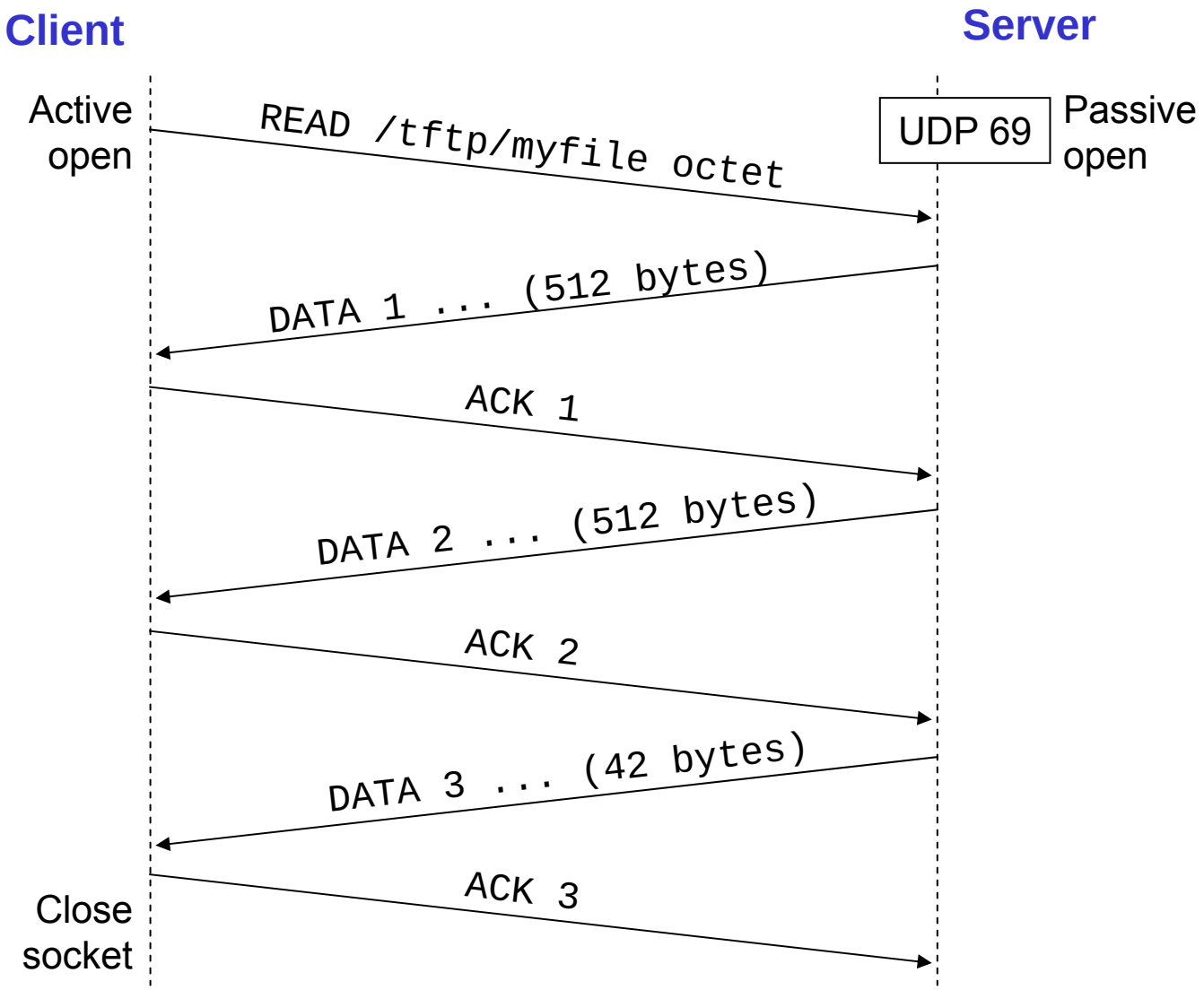
- Files are broken up into blocks of 512 bytes each.
- Last block has 0 – 511 bytes, thus a block of less than 512 bytes indicates the end of the file.
- Blocks have 16 bit sequence numbers, thus 32 MB maximum.

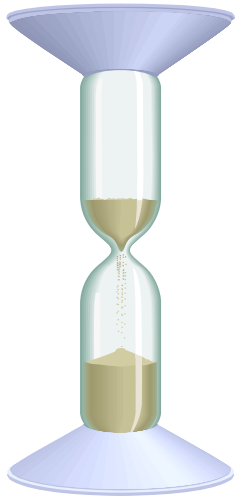
- Note: Meanwhile, the original TFTP protocol (RFC 783) has been revised to overcome some of these limitations.

TFTP Message Formats



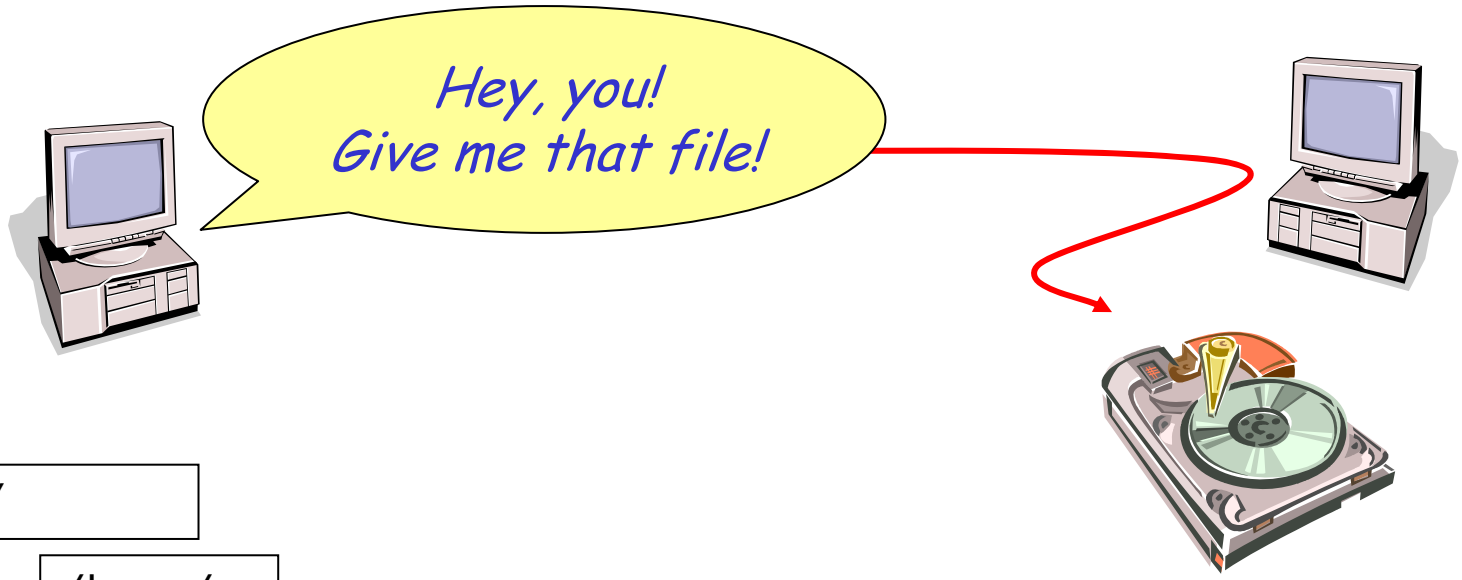
TFTP Example





Making the IP layer work …

Addressing and Naming



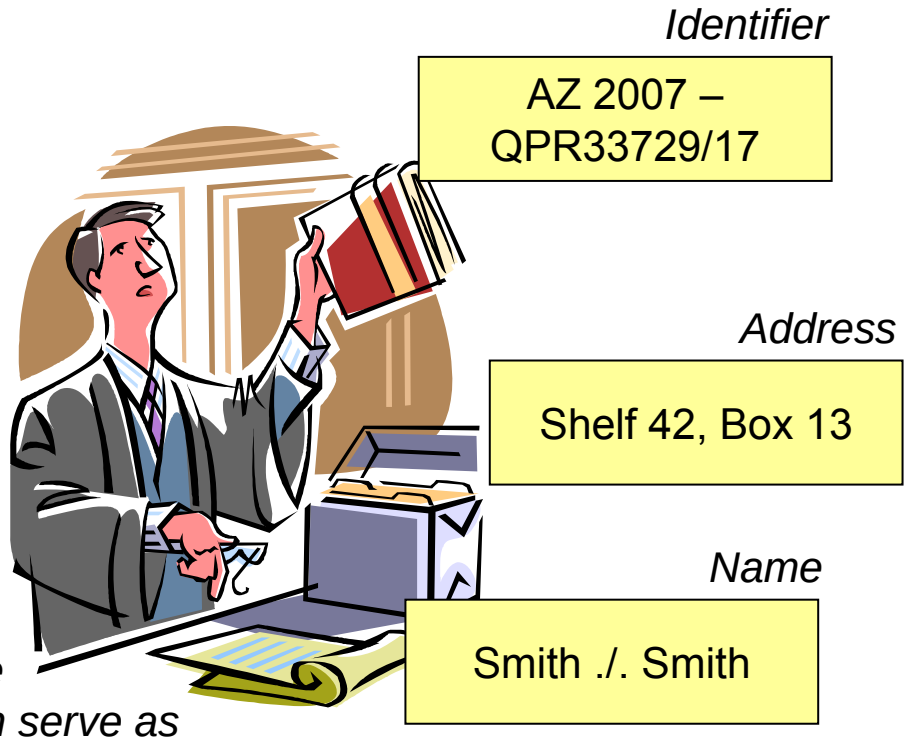
- /
  - /home/
    - /home/fuhrmann
    - /home/smith
  - /usr/
  - /var/

A file system maps human readable names to sectors on a hard disk.

The domain name system (DNS) maps human readable host names to IP addresses.

Identifier, Address, Name

- Identifier
  - System wide unique
- Address
  - Physical identifier
- Name
  - Logical identifier
  - Human readable



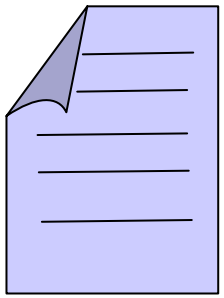
Note that typically both the address and the name can serve as identifier because they are unique!

Neither of these criteria is generally accepted.
There is no exact definition of the terms.

Network Configuration Files

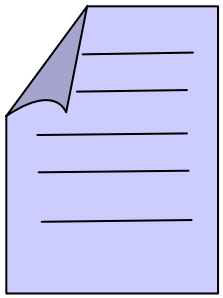
How does a host know the IP addresses of the (other) hosts in the network?

`/etc/hosts`



List of host names with according IP addresses.

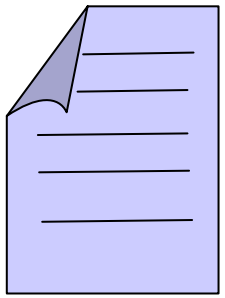
`/etc/sysconfig/…`



IP addresses of local interfaces, default router, …

⇔ ***Boot scripts***

`/etc/resolv.conf`

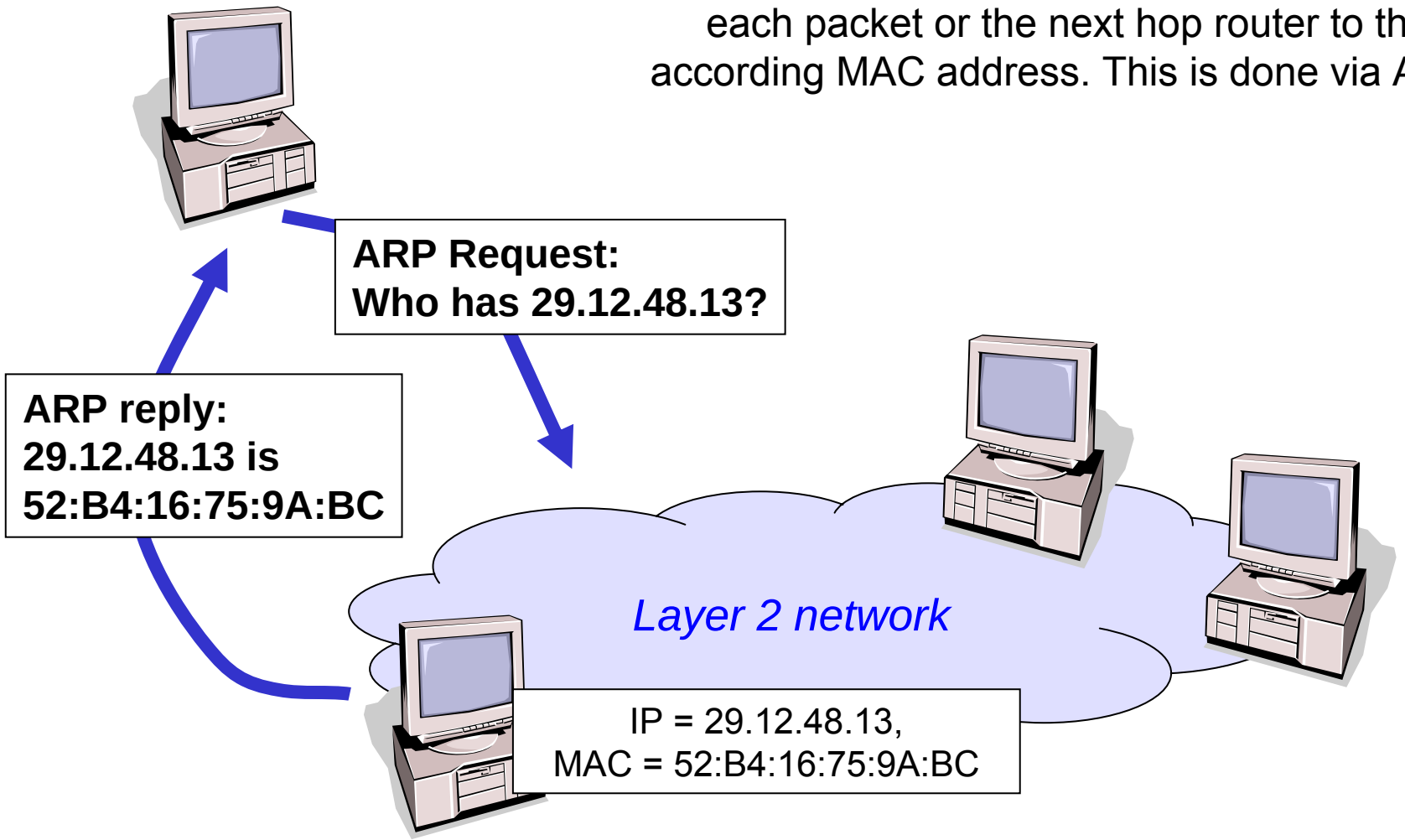


Domain name server

⇔ ***Boot scripts, DHCP***

Recap: Address Resolution

Internet hosts need to resolve the IP address of each packet or the next hop router to the according MAC address. This is done via ARP.

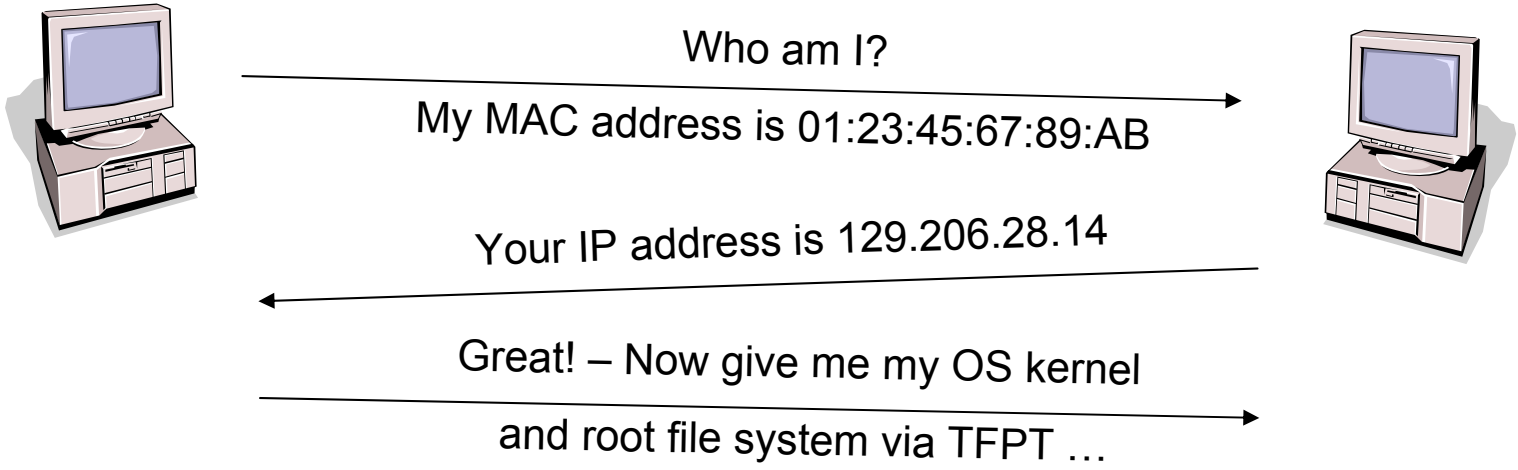


ARP Packet Format

Hardware type e.g. Ethernet = 1		Protocol type, e.g. IPv4 = 2048
Hardware length e.g. Ethernet = 6	Protocol length e.g. IPv4 = 4	Operation 1 = request, 2 = reply
Sender hardware address		
Sender protocol address		
Target hardware address (set to zero in request)		
Target protocol address		

Note: Of course, ARP is carried in a data link layer datagram such as an Ethernet frame!

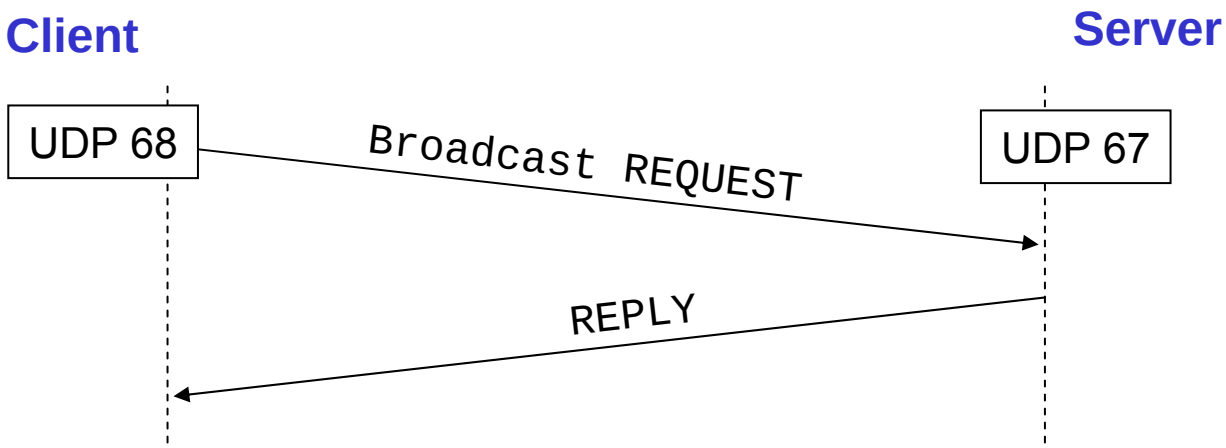
Reverse ARP



- Reverse ARP (=RARP) uses the ARP packet format to do an inverse mapping of hardware address to network layer address.
- RARP was designed to boot diskless computers from the network.
- RARP clients assume that the host of the RARP server that responded to their request also runs a TFTP server.
- After having received an image of the OS kernel and root file system, everything can proceed “as usual”.
- Note that still every host needs to be configured, namely by providing a per host file system image on the TFTP server. (The images are named by the hex string of the IP address.)

BOOTP

- BOOTP (=bootstrap protocol, RFC 951) replaces RARP and provides extensive functionality to boot diskless computers.
- It uses UDP/IP packets to carry messages, but hosts are still identified by their MAC address.
- Request uses special case IP address 255.255.255.255 (=limited broadcast).
- Reply normally uses client's IP address, but may be broadcast. Hence the well-known client port.
- Messages are not forwarded by routers, but optional proxies may forward BOOTP messages.
- Note: Request transmission uses random timeout to avoid synchronization.



BOOTP Message Format (1)

Opcode 1 = Request, 2 = Reply	Hardware type 1 = Ethernet	Hardware address length	Hop Count
Transaction ID (→ set by the client and repeated by the server to match replies to requests)			
Number of seconds (... the client has been trying to bootstrap)		unused	
Client IP address (→ set by the client if it knows its IP address, otherwise zero)			
Your IP address (→ set by the server to the client's actual IP address)			
Server IP address (→ set by the server to the server's actual IP address)			
Gateway IP address (→ set by the proxy server, if any)			
Client hardware address (→ set by the client to simplify processing at the server, 16 bytes)			

BOOTP Message Format (2)

Server host name (→ optional null terminate string; 64 bytes)
Boot file name (→ optional null terminate string; 128 bytes)
Options (64 bytes)

BOOTP – Chicken / Egg Issues

How can the server send an IP datagram to the client, if the client doesn't know its own IP address (yet)? Whenever a bootreply is being sent, the transmitting machine performs the following operations:

1. If the client knows its own IP address ('client IP addr' field is nonzero), then the IP can be sent 'as normal', since the client will respond to ARPs.
2. If the client does not yet know its IP address ('client IP addr' zero), then the client cannot respond to ARPs sent by the transmitter of the bootreply.

There are two options for the solution:

a. If the transmitter has the necessary kernel or driver hooks to 'manually' construct an ARP address cache entry, then it can fill in an entry using the 'client hardware addr' and 'your IP addr' fields.

b. If the transmitter lacks these kernel hooks, it can simply send the bootreply to the IP broadcast address on the appropriate interface. This is only one additional broadcast over the previous case.

Cited from RFC 951

IP Broadcast

- Limited Broadcast
  - IP packets with destination address 255.255.255.255 are delivered to all hosts in the local subnet.
  - Typically, the link layer provides a broadcast mechanism so that IP broadcasts can be handled efficiently.
  - Note: If a host has multiple interfaces, the broadcast is often only sent via the first interface.
- Network broadcast
  - IP packets where the host part of the address is all one bits are sent to the respective network and broadcast there.
  - Example: Packets with destination 141.3.255.255 are routed to the network 141.3.0.0 and broadcast there.
  - Note that this mechanism is obsolete due to security reasons ($\rightarrow$multicast).

Further Notes on Addresses

- The special address 0.0.0.0 denotes the „unknown address“. It is used as source address when a host has not yet obtained a valid address.
- A host part with all zeros denotes the respective network. Example: 141.3.0.0 is a network, 141.3.0.1 is a machine in that network.
- The network 127.0.0.0 is the “loop back” network. All traffic destined to that network is delivered at the local machine. Typically, 127.0.0.1 is used to denote the “local host”.
- The networks 10.0.0.0 and 192.168.x.0 are private networks, i.e. public Internet routers should drop all packets with these addresses.

BOOTP Shortcomings

- BOOTP was designed for relatively static environments where each host has a permanent network connection
  - The site's network administrators create a BOOTP configuration file with the parameters for each host.
  - This is only worth while when these files are typically stable for long periods.
- Dial-up hosts and wireless hosts are much more dynamic.
  - Network administrators want to reserve a pool of IP addresses for these hosts.
  - Upon joining the network, hosts are assigned one of these addresses.
  - After a node leaving the network, its address may be re-used.
- Dynamic address assignment is typical today, but it breaks the original Internet spirit where IP addresses identify hosts.

Dynamic Host Configuration Protocol

DHCP extends BOOTP for more dynamic host configuration:

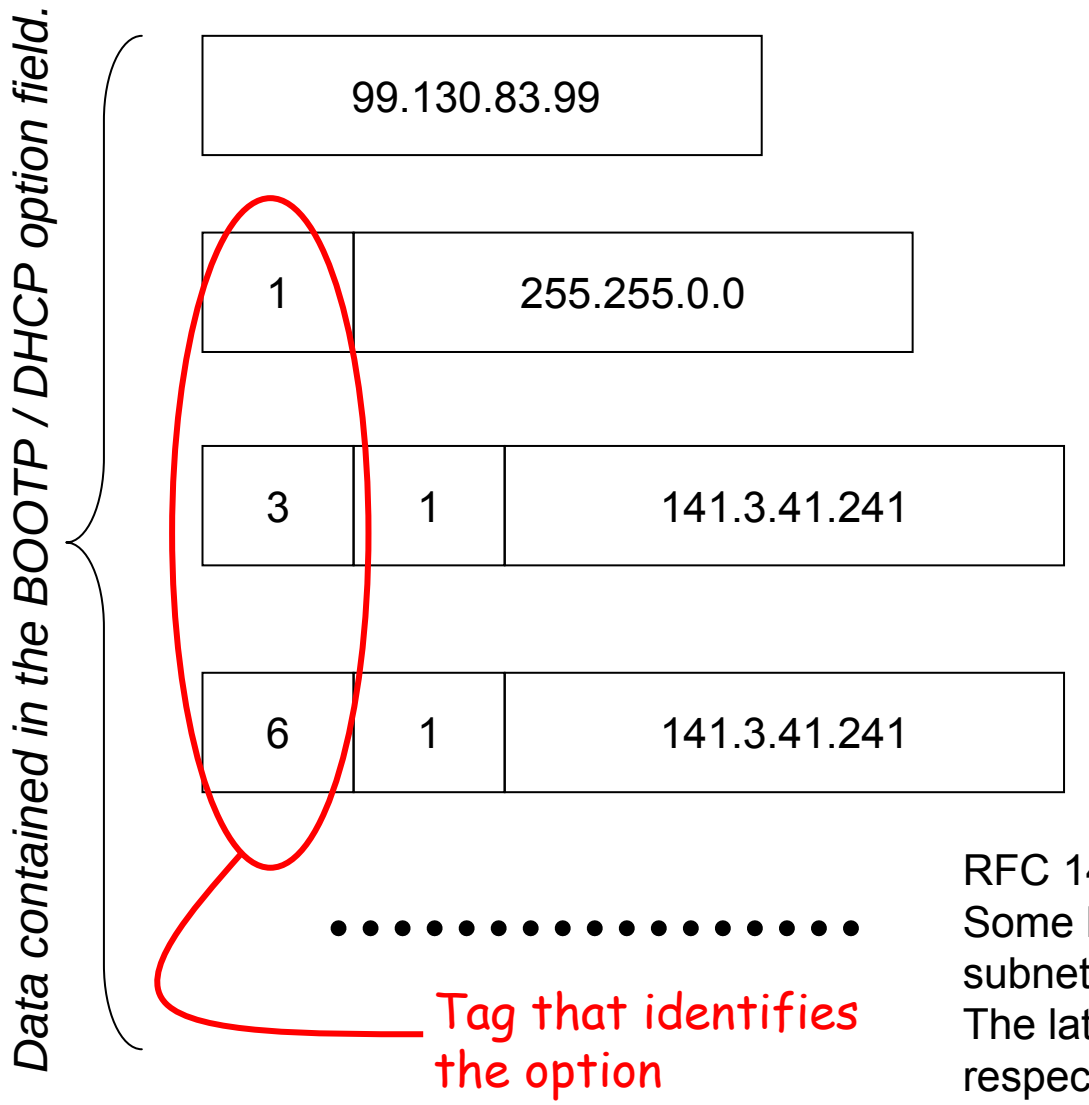
- Manual allocation – The DHCP server hands out IP addresses based on a table of MAC addresses. This table needs to be set up by the network administrator.
- Automatic allocation – The DHCP server automatically chooses a free IP address from the range that it was given by the network administrator. This address then permanently belongs to the respective client.
- Dynamic allocation – Like automatic allocation, but the address is associated with a finite lease time after which the server may assign the address to a different client.

With dynamic allocation addresses expire so that DHCP can deal with ungracefully leaving clients.

- While staying in the network clients need to regularly refresh their lease.

Like with BOOTP, options convey the addresses of DNS servers, etc.

BOOTP / DHCP Options (RFC 1497)



Magic number, used to identify a RFC conformant option list.

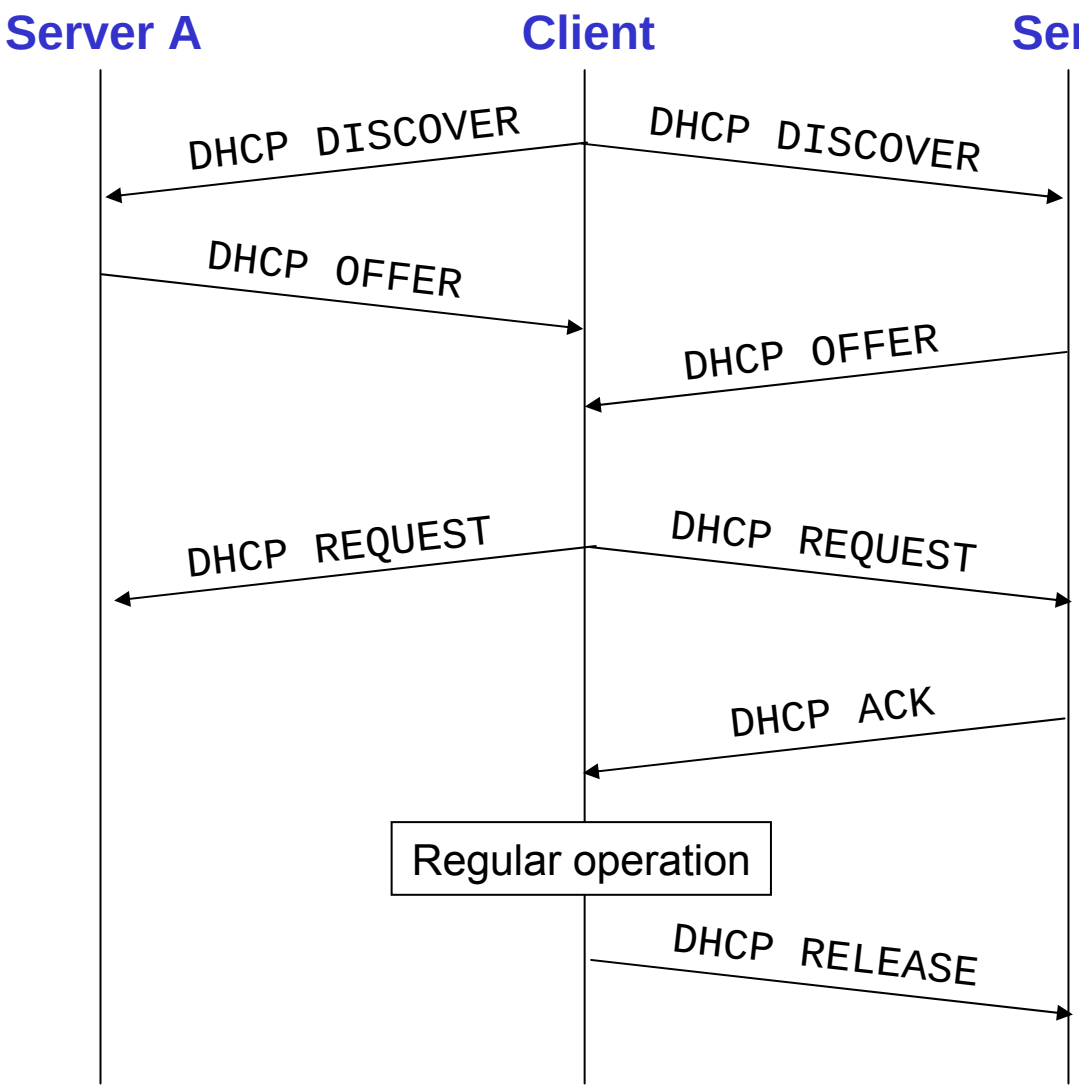
Subnet mask option

List of routers option

List of DNS servers option

RFC 1497 and others list many options. Some have fixed length such as the subnet mask, some have variable length. The latter indicate the length of the respective list immediately after the tag.

DHCP Operation (RFC 1531)

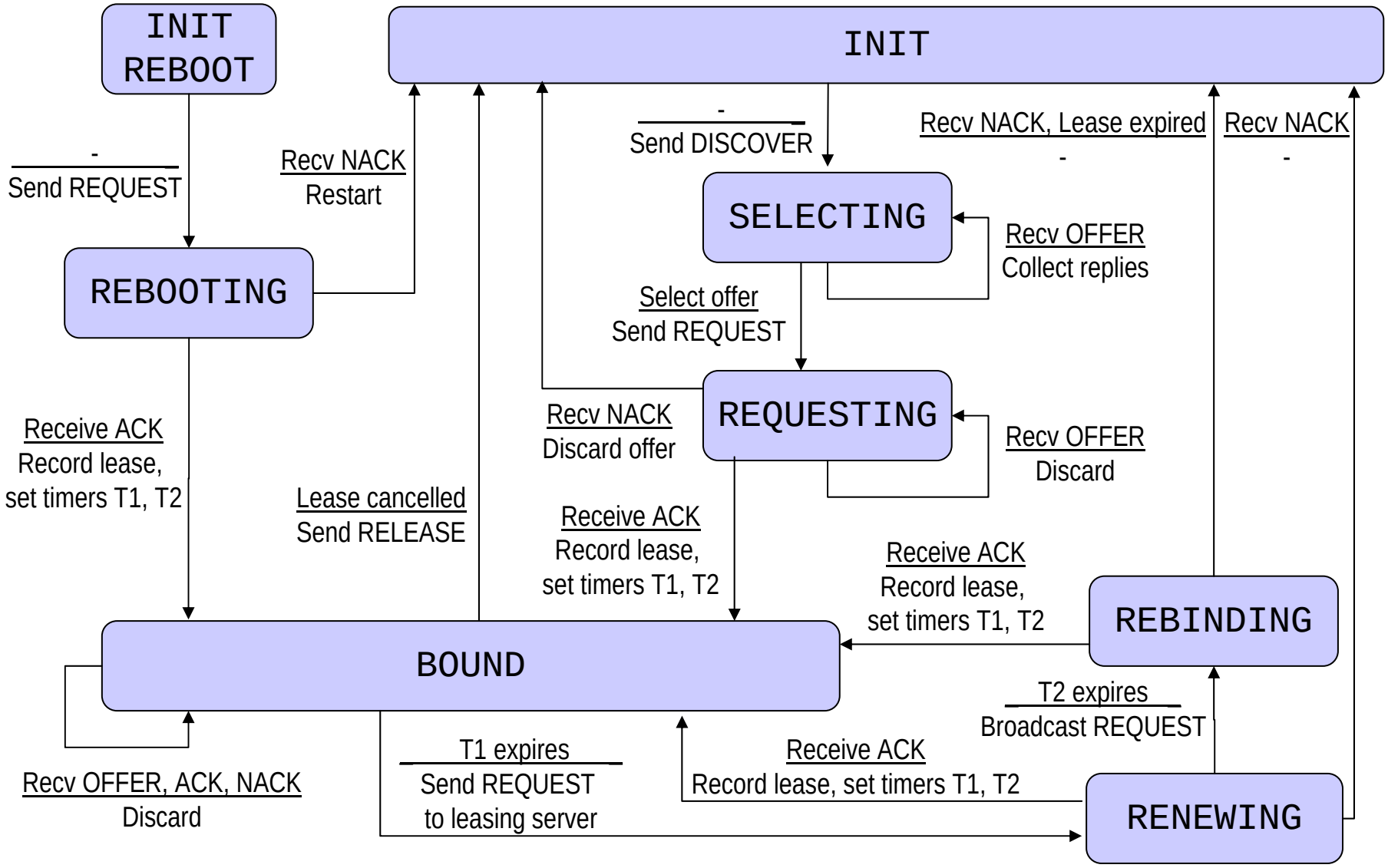


1. Client broadcasts message to discover servers.
2. Servers determine configuration and offer available addresses.
3. Client collects replies and requests selected configuration.
4. Server acknowledges request.
5. Client releases address. Server may keep record of binding.

DHCP Message Overview

- DHCPDISCOVER - Client broadcast to locate available servers.
- DHCPOFFER - Server to client in response to DHCPDISCOVER with offer of configuration parameters.
- DHCPREQUEST - Client message to servers either (a) requesting offered parameters from one server and implicitly declining offers from all others, (b) confirming correctness of previously allocated address after, e.g., system reboot, or (c) extending the lease on a particular network address.
- DHCPACK - Server to client with configuration parameters, including committed network address.
- DHCPNAK - Server to client indicating client's notion of network address is incorrect (e.g., client has moved to new subnet) or client's lease as expired
- DHCPDECLINE - Client to server indicating network address is already in use.
- DHCPRELEASE - Client to server relinquishing network address and cancelling remaining lease.
- DHCPINFORM - Client to server, asking only for local configuration parameters; client already has externally configured network address.

DHCP State Diagramm



Questions?

Thomas Fuhrmann

Department of Informatics
Self-Organizing Systems Group
c/o I8 Network Architectures and Services
Technical University Munich, Germany

fuhrmann@net.in.tum.de