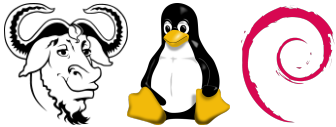
TFTP und der Zauberlehrling

Eine Einführung in Netzprotokolle

Andreas B. Mundt
andi@debian.org

Chemnitzer Linuxtage 2025

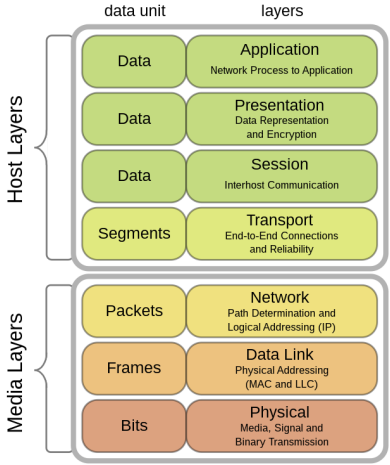
23. März 2025



Ausblick

1. Wie werden Daten im Netz übertragen?
   - Layer 2: Ethernet Protokoll
   - Layer 3: Internet Protokoll
   - Layer 4: Prozesse verbinden
2. Das TFTProtocol
   - TFTP Anwendung
   - Der Zauberlehrling und weitere Defizite
   - RFCs: Requests for Comments
   - TFTP Option Extension
3. Das Transmission Control Protocol (TCP)
   - Buchhaltung
   - `netcat`: TCP-Verbindung
   - Ausblick TCP

Das ISO/OSI-Referenzmodell

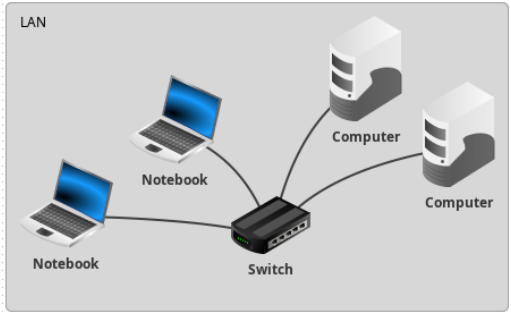


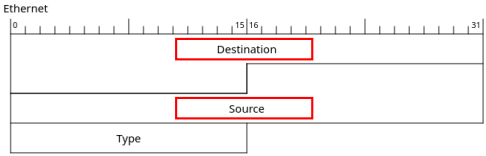
- Physical und Data Link Layer → Daten an das nächste Gerät im LAN senden
- Network Layer (IP) → Rechner in verschiedenen Netzen adressieren
- Transport Layer (UDP, TCP, . . .) → Prozesse auf verschiedenen Rechnern verbinden

Im einfachsten Fall (**Trivial** FTP) mit dem User Datagram Protocol (UDP)

Layer 2: Ethernet Protokoll

Daten an das nächste Gerät im LAN senden:



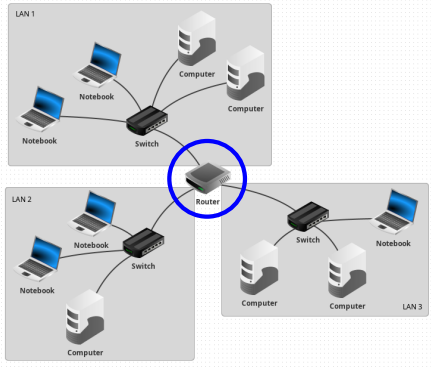


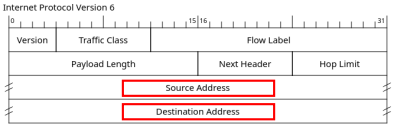
- Jedes Netzwerk-Interface hat eine MAC-Adresse (48 Bit)
- MAC Adressen sind eindeutig, haben aber keine topologische Struktur:
  - Organizationally Unique Identifier (OUI, 24 Bit)
  - Seriennummer (24 Bit)

  $\rightarrow$ nicht routingfähig
- Damit können Geräte im selben LAN (Netzwerk) miteinander kommunizieren

Layer 3: Netze verbinden

Daten in andere LANs senden:





- Jedes Interface hat eine IP-Adresse
- IP-Adressen haben eine topologische Struktur
- Router verbinden LANs: Sie haben Interfaces in verschiedenen Netzen und wissen „in welcher Richtung“ welche IP-Adress-Blöcke liegen

→ routingfähige Struktur

- Daten können damit zwischen beliebigen Geräten im Internet ausgetauscht werden

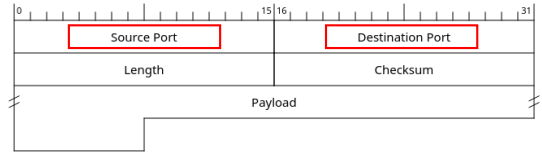
Layer 4: Prozesse verbinden

Eigentlich wollen wir Daten zwischen Prozessen austauschen.

Dazu benötigen wir noch eine „Adresse“, die sog. **Portnummer**, damit das Betriebssystem Daten Prozessen zuordnen kann.

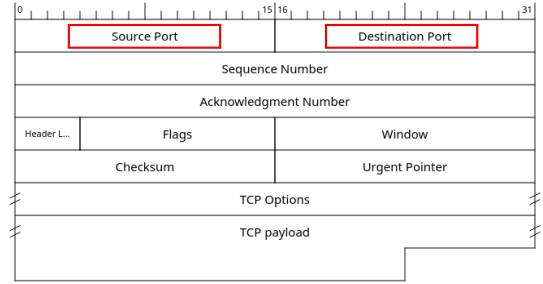
Mehrere Varianten existieren, z.B. das *User Datagram Protocol (UDP)* oder das *Transmission Control Protocol (TCP)*.

User Datagram Protocol

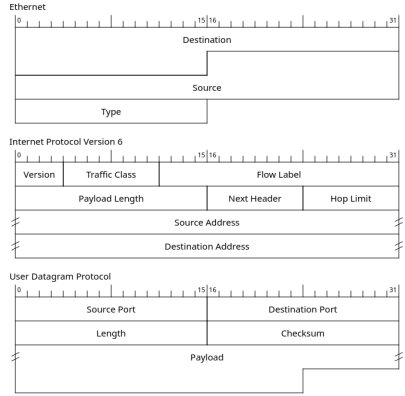


Oder:

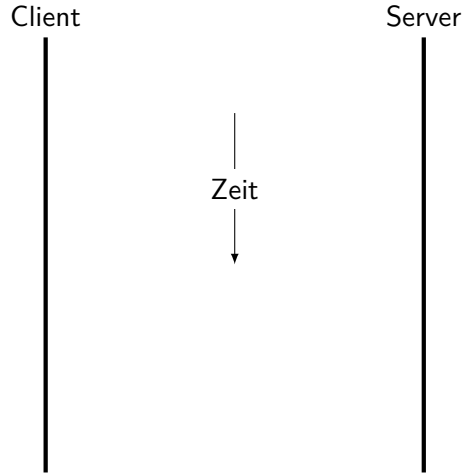
Transmission Control Protocol



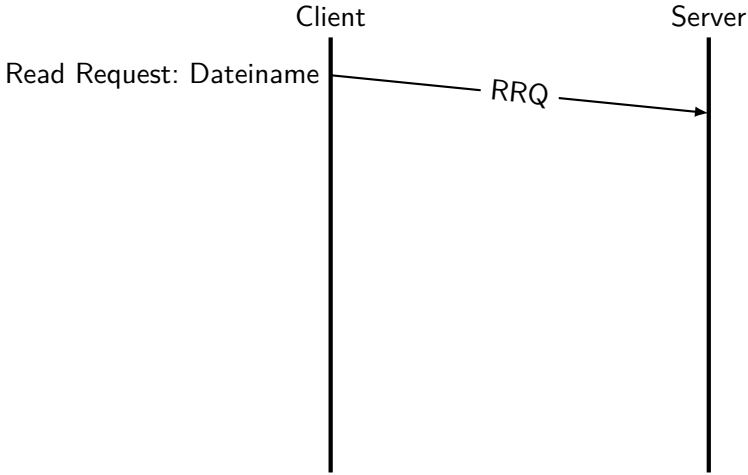
Datenübertragung mittels UDP



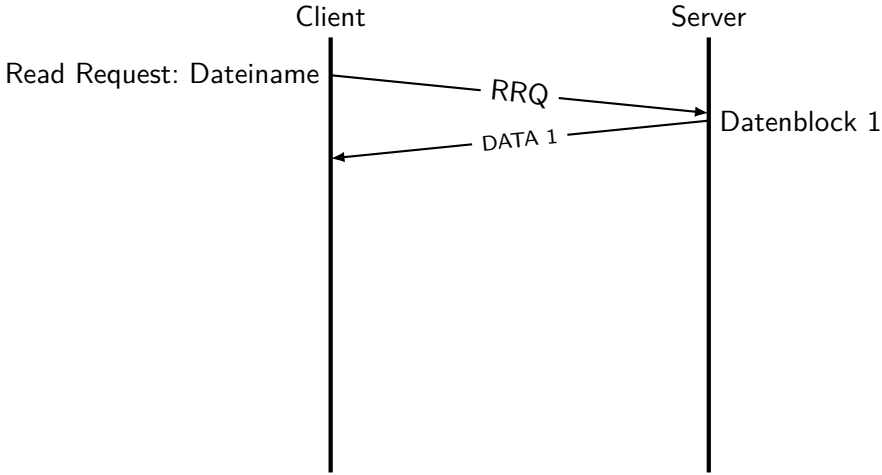
Trivial File Transfer Protocol



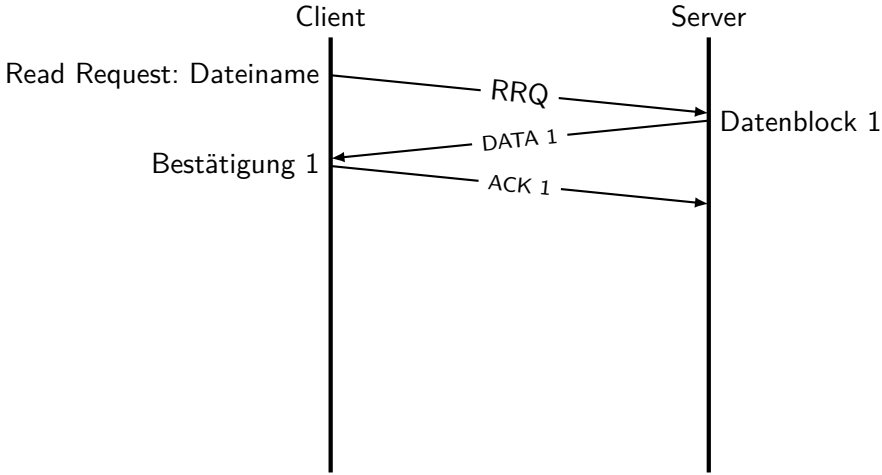
Trivial File Transfer Protocol



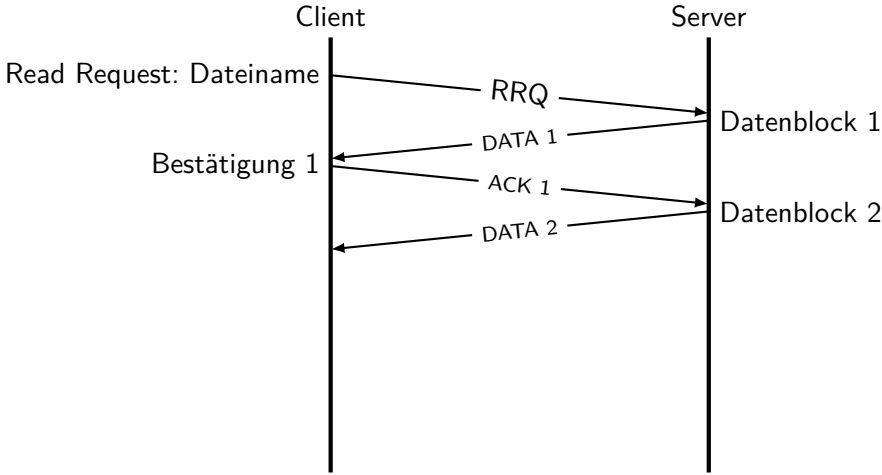
Trivial File Transfer Protocol



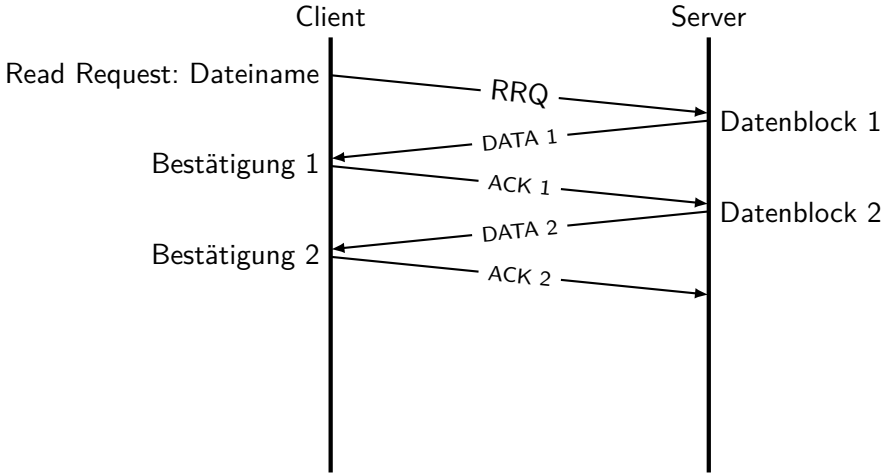
Trivial File Transfer Protocol



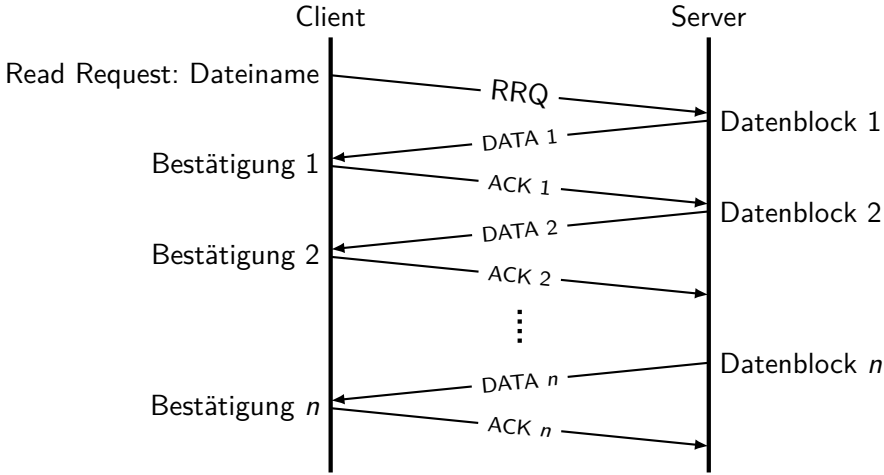
Trivial File Transfer Protocol



Trivial File Transfer Protocol



Trivial File Transfer Protocol



TFTP Algorithmus (ursprünglich)

Server:

- Nach dem Read Request RRQ: Sende Datenblock DATA 1
- Sende für Bestätigung ACK j den nächsten[1] Datenblock DATA $j + 1$

Client:

- Bestätige jeden Datenblock k mit dem zugehörigen Acknowledgement k
- Der letzte Datenblock ist kleiner als die vorhergehenden oder ganz leer.

Was, wenn eine „Antwort“ (DATA oder ACK) ausbleibt[2]?

$\rightarrow$ Sende nach Timeout einfach das letzte Datagram erneut.

[1] solange $j \neq n$

[2] z.B. durch Paketverlust

TFTP Algorithmus (ursprünglich)

Server:

- Nach dem Read Request RRQ: Sende Datenblock DATA 1
- Sende für Bestätigung ACK j den nächsten[1] Datenblock DATA $j + 1$

Client:

- Bestätige jeden Datenblock k mit dem zugehörigen Acknowledgement k
- Der letzte Datenblock ist kleiner als die vorhergehenden oder ganz leer.

Was, wenn eine „Antwort“ (DATA oder ACK) ausbleibt[2]?

→ Sende nach Timeout einfach das letzte Datagram erneut.

[1] solange $j \neq n$

[2] z.B. durch Paketverlust

TFTP Algorithmus (ursprünglich)

Server:

- Nach dem Read Request RRQ: Sende Datenblock DATA 1
- Sende für Bestätigung ACK j den nächsten[1] Datenblock DATA $j + 1$

Client:

- Bestätige jeden Datenblock k mit dem zugehörigen Acknowledgement k
- Der letzte Datenblock ist kleiner als die vorhergehenden oder ganz leer.

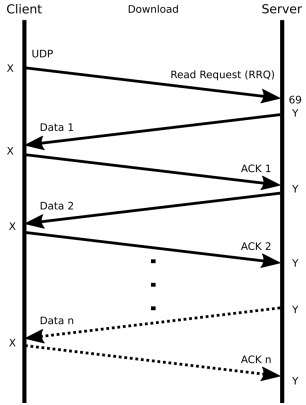
Was, wenn eine „Antwort“ (DATA oder ACK) ausbleibt[2]?

$\rightarrow$ Sende nach Timeout einfach das letzte Datagram erneut.

[1] solange $j \neq n$

[2] z.B. durch Paketverlust

TFTP Anwendung



No.	Time	Source	Source Port	Destination	Destination Port	Protocol	Length	Info
1	0.000000000	::1	41944	::1	69	TFTP	93	Read Request, File: d-i/n-
2	0.021296925	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 1
3	0.021360085	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 1
4	0.021398777	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 2
5	0.021429127	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 2
6	0.021469999	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 3
7	0.021498167	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 3
8	0.021527434	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 4
9	0.021556974	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 4
10	0.021573057	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 5
11	0.021604702	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 5
12	0.021619356	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 6
13	0.021631370	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 6
14	0.021644470	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 7
15	0.021656429	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 7
16	0.021669326	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 8
17	0.021681239	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 8
18	0.021697141	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 9
19	0.021744167	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 9
20	0.021758924	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 10
21	0.021772046	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 10
3645	0.061388936	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 1822
3646	0.061398571	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 1823
3647	0.061407236	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 1823
3648	0.061416838	::1	54995	::1	41944	TFTP	578	Data Packet, Block: 1824
3649	0.061425553	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 1824
3650	0.061437327	::1	54995	::1	41944	TFTP	418	Data Packet, Block: 1825 (last)
3651	0.061454647	::1	41944	::1	54995	TFTP	66	Acknowledgement, Block: 1825

[2] Z.B.: `atftpd --daemon --trace --no-fork --verbose=7 --logfile=/dev/stdout --port=6969 /tmp/`
`atftp --get --remote-file TFTP.toc --local-file /dev/null 127.0.0.1 6969 --trace --option "blksize 256"`

TFTP Anwendung

Anwendungsbereiche TFTP:

- Aufgrund der Einfachheit nur im lokalen Netz (LAN)
- Booten über das Netzwerk, Preboot Execution Environment (PXE)
- Embedded Systems: Firmware Images flashen auf Geräte wie Router, Firewalls, IP Phones, ...

Freie Open-Source TFTP-Server:

- `tftpd-hpa`: HPA's tftp server
- `atftpd`: advanced TFTP server
- `dnsmasq`: small caching DNS proxy and DHCP/TFTP server
- ...

TFTP Anwendung

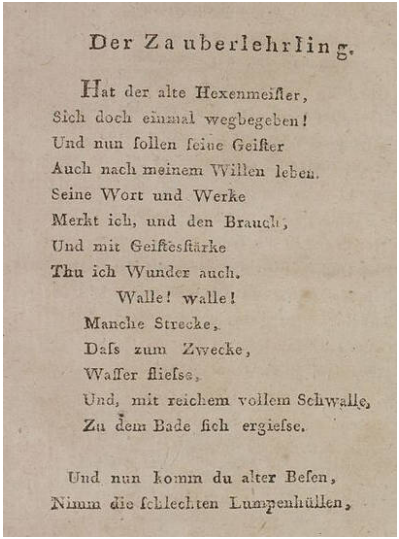
Anwendungsbereiche TFTP:

- Aufgrund der Einfachheit nur im lokalen Netz (LAN)
- Booten über das Netzwerk, Preboot Execution Environment (PXE)
- Embedded Systems: Firmware Images flashen auf Geräte wie Router, Firewalls, IP Phones, ...

Freie Open-Source TFTP-Server:

- `tftpd-hpa`: HPA's tftp server
- `atftpd`: advanced TFTP server
- `dnsmasq`: small caching DNS proxy and DHCP/TFTP server
- ...

Der Zauberlehrling und weitere Defizite



Der Zauberlehrling.

Hat der alte Hexenmeister,
Sich doch einmal wegbegeben!
Und nun sollen seine Geister
Auch nach meinem Willen leben.
Seine Wort und Werke
Merkt ich, und den Brauch,
Und mit Geistesstärke
Thu ich Wunder auch.
Walle! walle!
Manche Strecke,
Dass zum Zwecke,
Wasser fliesse,
Und, mit reichem vollem Schwalle,
Zu dem Bade sich ergiesse.

Und nun komm du alter Besen,
Nimm die schlechten Lumpenhüllen,

Sorcerer's Apprentice Syndrome

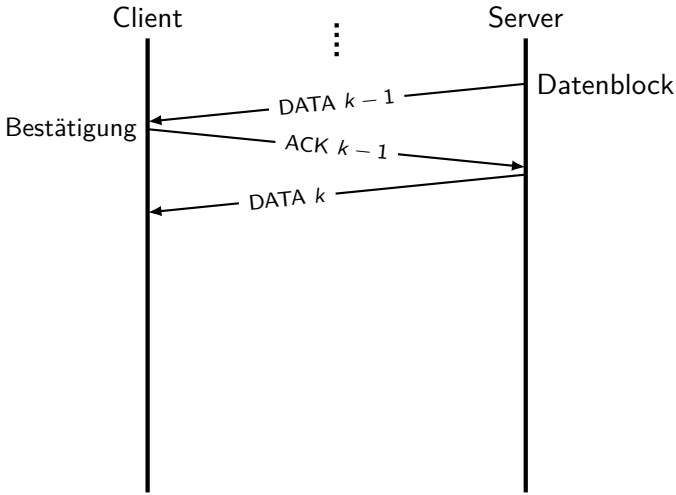
```
[...]
Seht da kommt er schleppend wieder!
Wie ich mich nur auf dich werfe,
gleich, o Kobold, liegst du nieder;
krachend trifft die glatte Schärfe.
Wahrlich, brav getroffen!
Seht, er ist entzwei!
Und nun kann ich hoffen,
und ich atme frei!

Wehe! wehe!
Beide Teile
stehn in Eile
schon als Knechte
völlig fertig in die Höhe!
Helft mir, ach! ihr hohen Mächte!
```

Sorcerer's Apprentice Syndrome

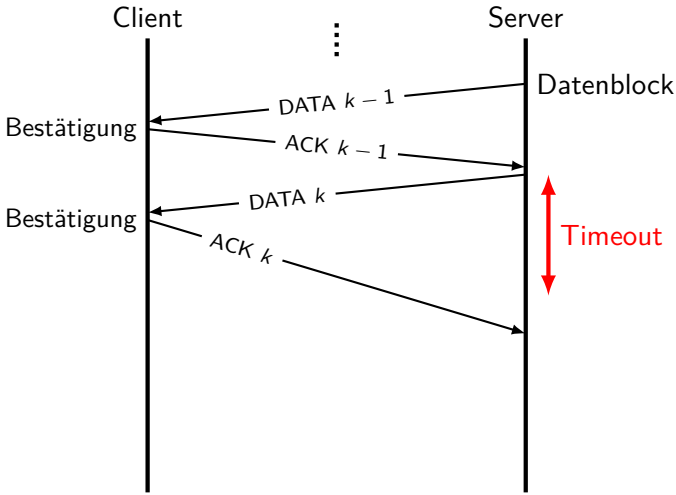
```
[...]
Seht da kommt er schleppend wieder!
Wie ich mich nur auf dich werfe,
gleich, o Kobold, liegst du nieder;
krachend trifft die glatte Schärfe.
Wahrlich, brav getroffen!
Seht, er ist entzwei!
Und nun kann ich hoffen,
und ich atme frei!

Wehe! wehe!
Beide Teile
stehn in Eile
schon als Knechte
völlig fertig in die Höhe!
Helft mir, ach! ihr hohen Mächte!
```

```
Und sie laufen! Naß und nässer
wirds im Saal und auf den Stufen.
Welch entsetzliches Gewässer!
Herr und Meister! hör mich rufen! -
Ach, da kommt der Meister!
Herr, die Not ist groß!
Die ich rief, die Geister
werd ich nun nicht los.
```

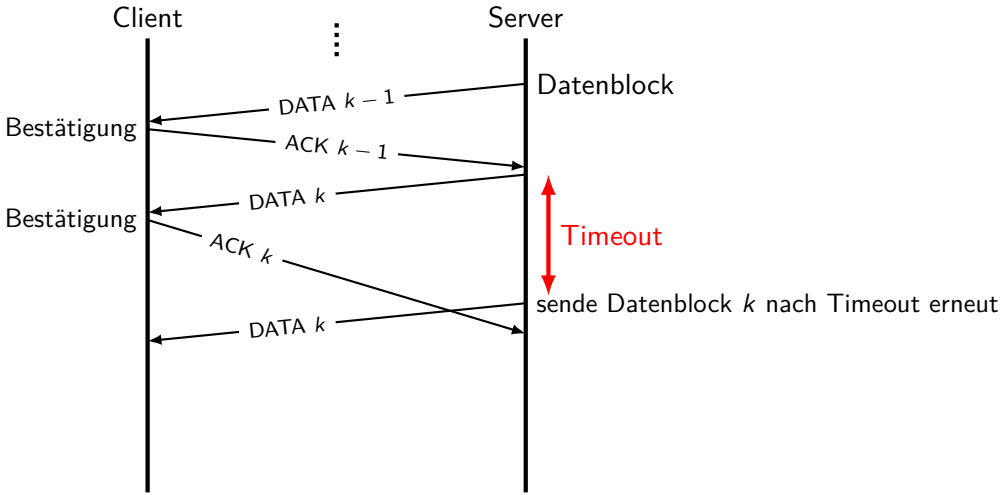
Sorcerer's Apprentice Syndrome



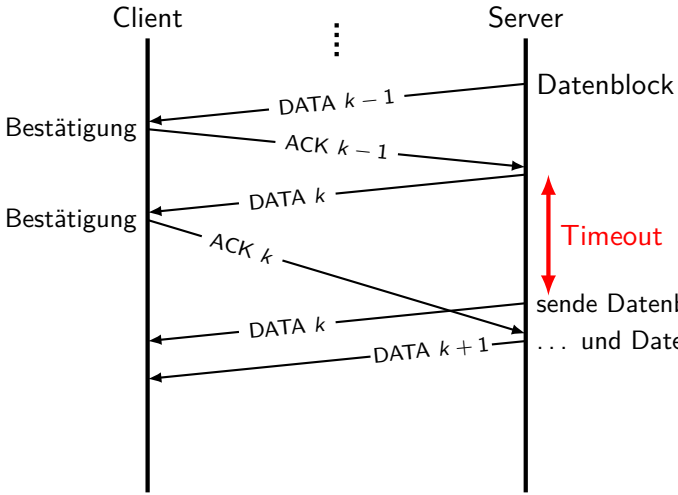
Sorcerer's Apprentice Syndrome



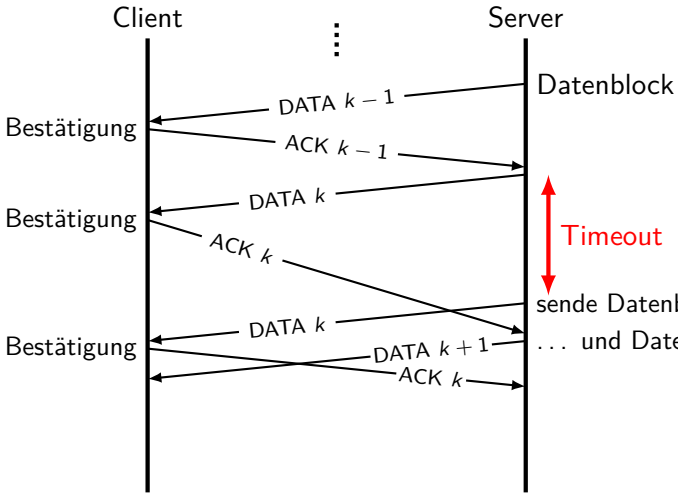
Sorcerer's Apprentice Syndrome



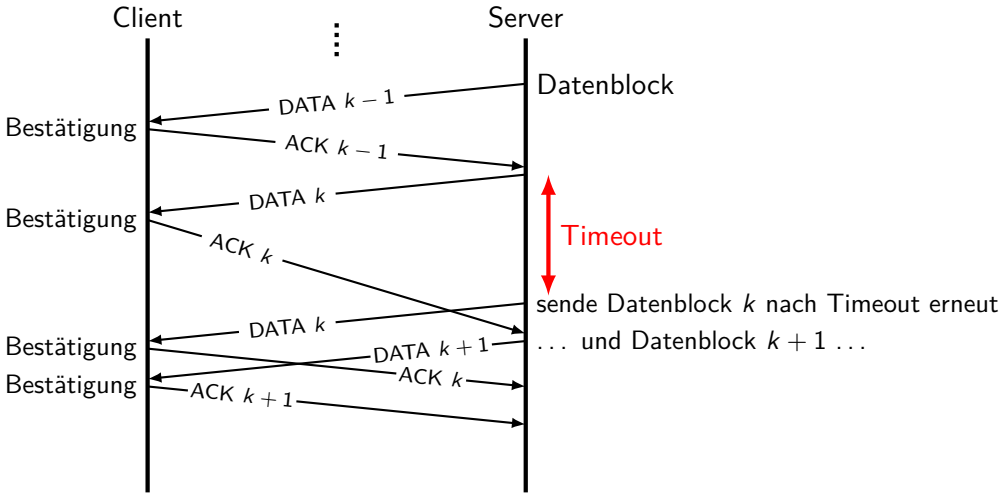
Sorcerer's Apprentice Syndrome



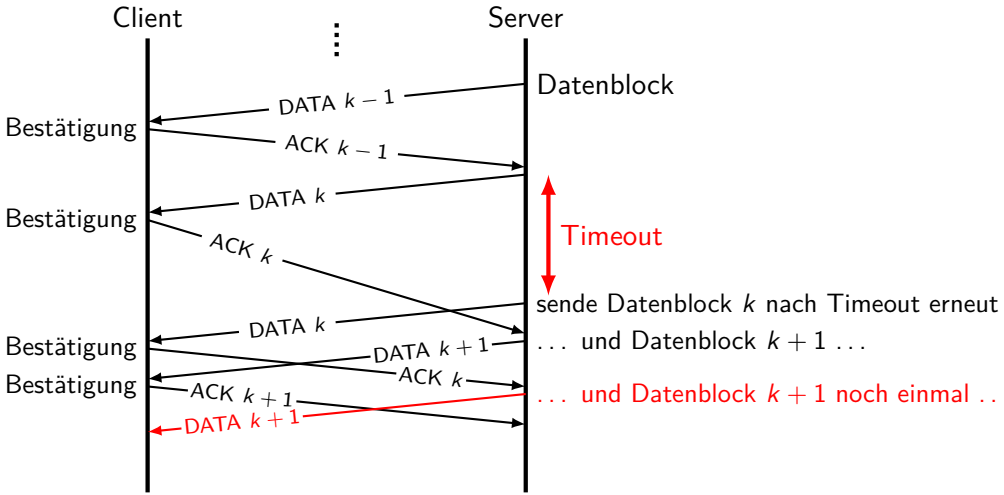
Sorcerer's Apprentice Syndrome



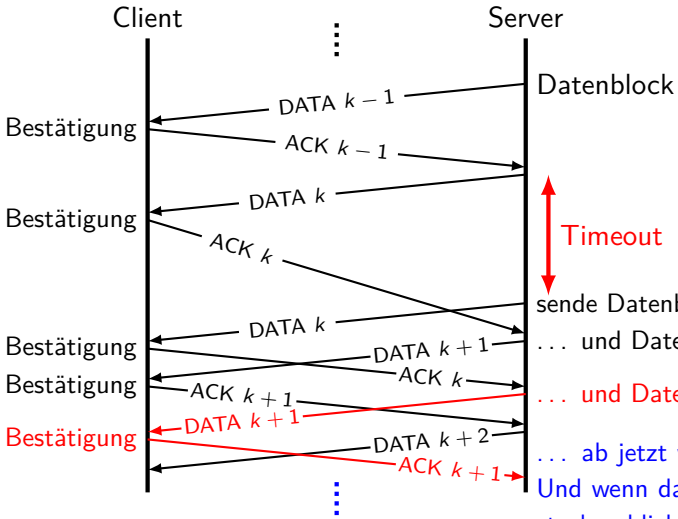
Sorcerer's Apprentice Syndrome



Sorcerer's Apprentice Syndrome

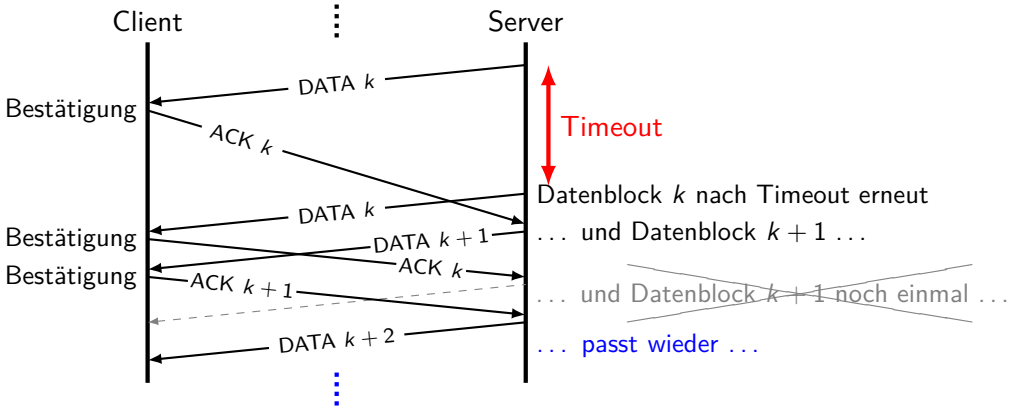


Sorcerer's Apprentice Syndrome



Sorcerer's Apprentice Syndrome

Lösung: Sende einen Datenblock nur einmal, es sei denn ein Timeout tritt auf.



Weitere Defizite des ursprünglichen TFTP

- Die Größe eines Datenblocks ist mit 512 Bytes fix vorgegeben.
- Der Empfänger einer Datei hat bis zum letzten Block keine Kenntnis über die insgesamt zu erwartende Datenmenge.
- Timeouts sind fix konfiguriert.
- Schlechte Performance: Pro Datenblock eine Round Trip Time (RTT).

Weitere Defizite des ursprünglichen TFTP

- Die Größe eines Datenblocks ist mit 512 Bytes fix vorgegeben.
- Der Empfänger einer Datei hat bis zum letzten Block keine Kenntnis über die insgesamt zu erwartende Datenmenge.
- Timeouts sind fix konfiguriert.
- Schlechte Performance: Pro Datenblock eine Round Trip Time (RTT).

Wie kann das Protokoll verbessert werden?

RFCs Trivial File Transfer Protocol

Request **F**or **C**omments:

- RFC 783 (First Revision, 1981)
- RFC 1123 (1989): *Sorcerer's Apprentice Syndrome*[3]
- RFC 1350 (Second Revision, 1992)
- RFC 2347 (TFTP Option Extension, 1998)
- RFC 2348 (TFTP Blocksize Option, 1998)
- RFC 2349 (TFTP Timeout Interval and Transfer Size Options, 1998)
- RFC 7440 (TFTP Windowsize Option, 2015)

[3] Cf. J.W. von Goethe, „Der Zauberlehrling“: „Die ich rief, die Geister werd ich nun nicht los.“ (1797)
(„The spirits that I summoned / I now cannot rid myself of again“)

RFC 2347: TFTP Option Extension (1998)

Der Client kann dem Server gewisse Parameter des Transfers vorschlagen/übermitteln:

- `tsize` → transfer size
- `blksize` → block size
- `timeout` → timeout :-)
- `windowsize` → Anzahl der „am Stück“ gesendeten Datenblöcke

Der Server kann diese dann bestätigen (OACK) oder bei den „defaults“ bleiben.

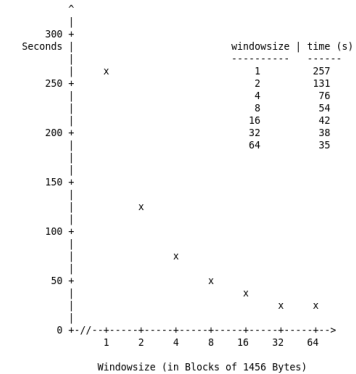
```
Read Request, File: d-i/n-a/menu.ipxe, Transfer type: octet, tsize=0, timeout=1, blksize=1434, windowsize=8
Option Acknowledgement, tsize=3593, timeout=1, blksize=1434, windowsize=8
Acknowledgement, Block: 0
Data Packet, Block: 1
Data Packet, Block: 2
Data Packet, Block: 3 (last)
Acknowledgement, Block: 3
```

TFTP Windowsize Option

RFC 7440 (2015)

```
[ DRCV ]      <---traffic--->      [ DSND ]
  ACK#       ->                <-   Data Block#   window block#
                     ...
                     <-            |DB n+01|          1
                     <-            |DB n+02|          2
                     <-            |DB n+03|          3
                     <-            |DB n+04|          4
|ACK n+04|          ->
                     <-            |DB n+05|          1
                Error |<-          |DB n+06|          2
                     <-            |DB n+07|          3
|ACK n+05|          ->
                     <-            |DB n+06|          1
                     <-            |DB n+07|          2
                     <-            |DB n+08|          3
                     <-            |DB n+09|          4
|ACK n+09|          ->
                     <-            |DB n+10|          1
                Error |<-          |DB n+11|          2
                     <-            |DB n+12|          3
|ACK n+10|          ->| Error
                     <-            |DB n+13|          4
                                 - timeout -
                     <-            |DB n+10|          1
                     <-            |DB n+11|          2
                     <-            |DB n+12|          3
                     <-            |DB n+13|          4
|ACK n+13|          ->
                     ...
```

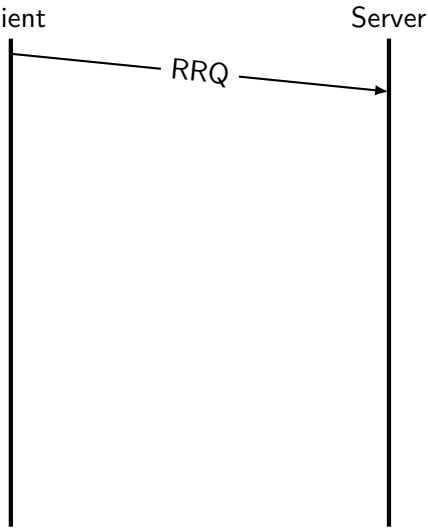
Performance tests were run on the prototype implementation using a variety of windowsizes and a fixed blocksize of 1456 bytes. The tests were run on a lightly loaded Gigabit Ethernet, between two Toshiba Tecra Core 2 Duo 2.2 Ghz laptops, in "octet" mode, transferring a 180 MByte file.



windowsize	time (s)
1	257
2	131
4	76
8	54
16	42
32	38
64	35

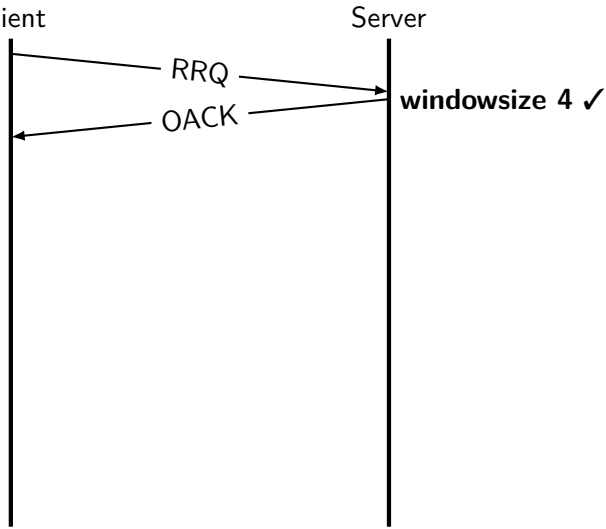
TFTP Windowsize Option

Read Request: Dateiname,
Option: **windowsize 4**

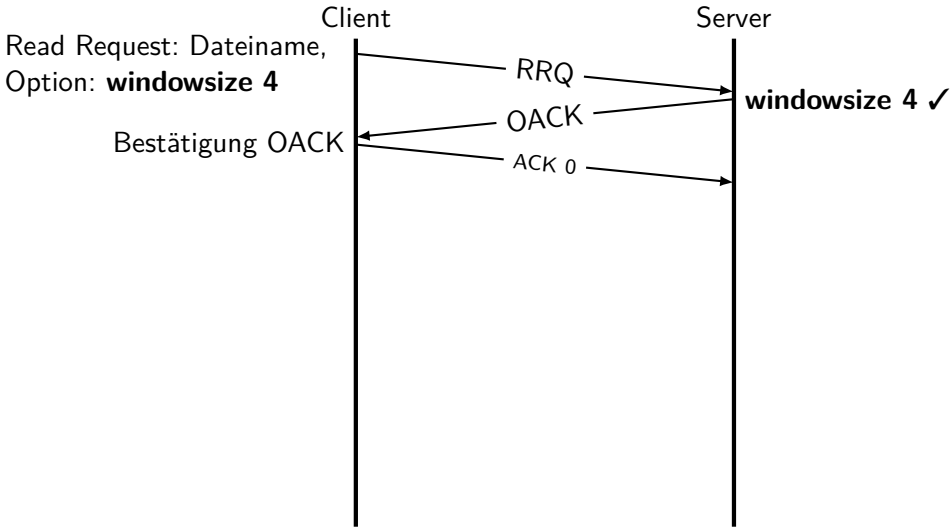


TFTP Windowsize Option

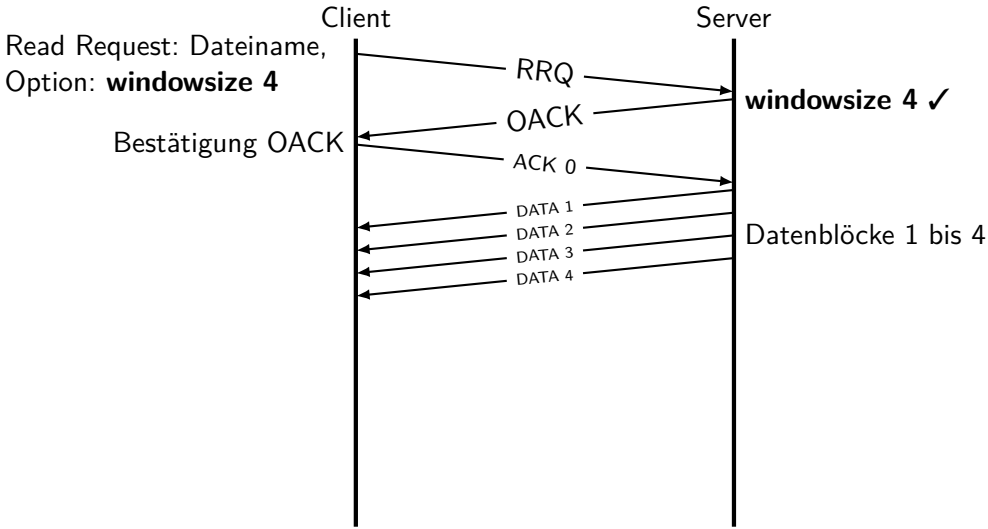
Read Request: Dateiname,
Option: **windowsize 4**



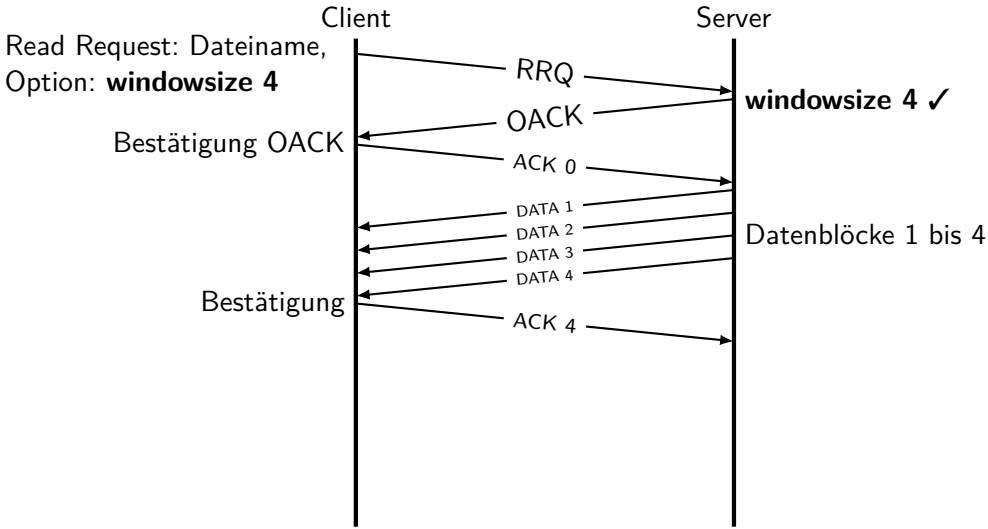
TFTP Windowsize Option



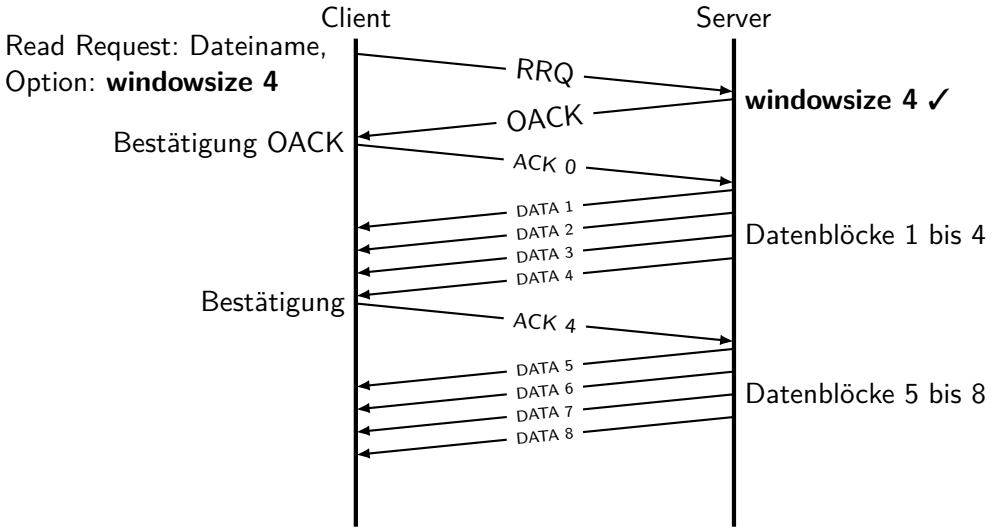
TFTP Windowsize Option



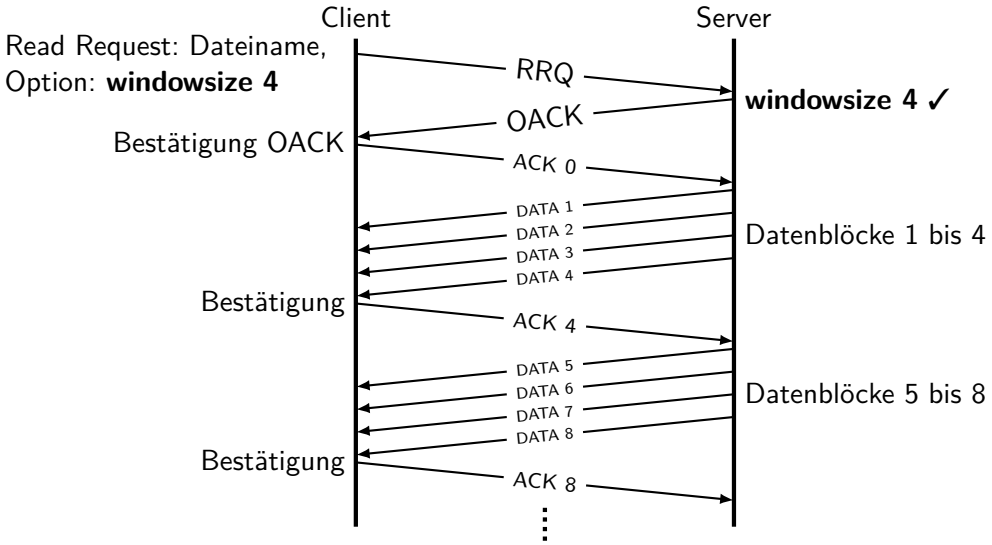
TFTP Windowsize Option



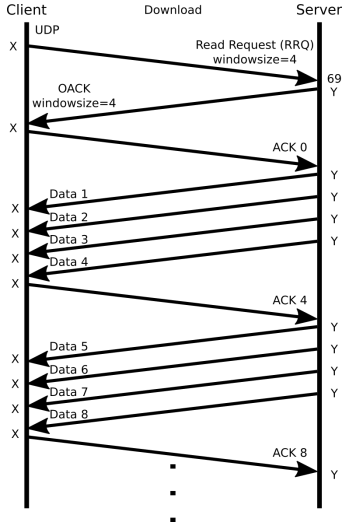
TFTP Windowsize Option



TFTP Windowsize Option



TFTP Windowsize Option



No.	Time	Source	Source Port	Destination	Destination Port	Protocol	Length	Info
1	0.000000000	::1	57394	::1	69	TFTP	101	Read Request, File: d-i/n-a/menu.ipx
2	0.019275714	::1	60556	::1	57394	TFTP	77	Option Acknowledgement, windowsize=4
3	0.019335292	::1	57394	::1	60556	TFTP	66	Acknowledgement, Block: 0
4	0.019597917	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 1
5	0.019614407	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 2
6	0.019622339	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 3
7	0.019629606	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 4
8	0.019651267	::1	57394	::1	60556	TFTP	66	Acknowledgement, Block: 4
9	0.019692844	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 5
10	0.019707647	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 6
11	0.019715360	::1	60556	::1	57394	TFTP	578	Data Packet, Block: 7
12	0.019724429	::1	60556	::1	57394	TFTP	75	Data Packet, Block: 8 (last)
13	0.019743221	::1	57394	::1	60556	TFTP	66	Acknowledgement, Block: 8
64	97.686022677	::1	54834	::1	69	TFTP	105	Read Request, File: d-i/n-a/grub/grub
65	97.703060307	::1	57246	::1	54834	TFTP	77	Option Acknowledgement, windowsize=4
66	97.703115588	::1	54834	::1	57246	TFTP	66	Acknowledgement, Block: 0
67	97.703368319	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 1
68	97.703383131	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 2
69	97.703390267	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 3
70	97.703396619	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 4
71	97.703415088	::1	54834	::1	57246	TFTP	66	Acknowledgement, Block: 4
72	97.703451215	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 5
73	97.703471965	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 6
74	97.703487414	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 7
75	97.703493439	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 8
76	97.703511357	::1	54834	::1	57246	TFTP	66	Acknowledgement, Block: 8
77	97.703540540	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 9
78	97.703558124	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 10
79	97.703563936	::1	57246	::1	54834	TFTP	578	Data Packet, Block: 11
80	97.703579999	::1	57246	::1	54834	TFTP	173	Data Packet, Block: 12 (last)
81	97.703646128	::1	54834	::1	57246	TFTP	66	Acknowledgement, Block: 12

```
▸ Frame 1: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ User Datagram Protocol, Src Port: 57394, Dst Port: 69
▾ Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: d-i/n-a/menu.ipxe
    Type: octet
  ▾ Option: windowsize = 4
      Option name: windowsize
      Option value: 4
```

Weitere Verbesserungen → TCP

Das Transmission Control Protocol (TCP)

Eine Verbesserung von TFTP adressiert u.a. die folgenden Problematiken:

- Wie kann man Packetverlusten oder -Verzögerung begegnen?
- Mit welchem zeitlichen Abstand sollte man die Datagramme innerhalb eines Fensters versenden?
- Congestion control?
- ...

Lösungsansatz:

- Um die Datenübertragung zu organisieren, benötigen wir Verwaltungsinformationen, einen „Zustand“ der Verbindung.
- Dieser wird erstmals über einen Verbindungsaufbau übertragen,
- dann während der Übertragung „mitgeführt“/„gepflegt“,
- und schließlich werden durch den Verbindungsabbau benutzte Ressourcen wieder freigegeben.

Das Transmission Control Protocol (TCP)

Eine Verbesserung von TFTP adressiert u.a. die folgenden Problematiken:

- Wie kann man Packetverlusten oder -Verzögerung begegnen?
- Mit welchem zeitlichen Abstand sollte man die Datagramme innerhalb eines Fensters versenden?
- Congestion control?
- ...

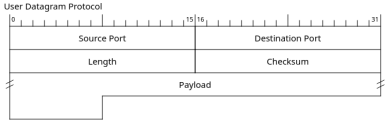
Lösungsansatz:

- Um die Datenübertragung zu organisieren, benötigen wir Verwaltungsinformationen, einen „Zustand“ der Verbindung.
- Dieser wird erstmals über einen Verbindungsaufbau übertragen,
- dann während der Übertragung „mitgeführt“/„gepflegt“,
- und schließlich werden durch den Verbindungsabbau benutzte Ressourcen wieder freigegeben.

UDP- und TCP-Header im Vergleich

UDP Protocol Header

- Source Port
- Destination Port
- Lenght
- Checksum



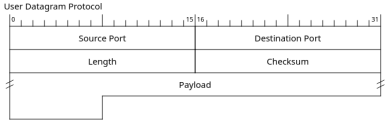
[1] In der Regel die Nummer des ersten Bytes der Payload, die im betreffenden Segment gesendet wird.

[2] In der Regel die Nummer des als nächstes erwarteten Bytes.

UDP- und TCP-Header im Vergleich

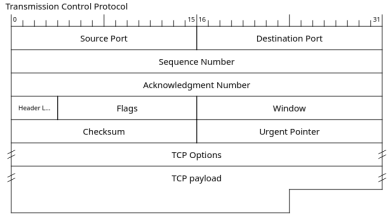
UDP Protocol Header

- Source Port
- Destination Port
- Lenght
- Checksum



TCP Protocol Header:

- Source Port
- Destination Port
- **Sequenz Number**[1]
- **Acknowledgment Number**[2]
- Flags, Checksum, Options, ...



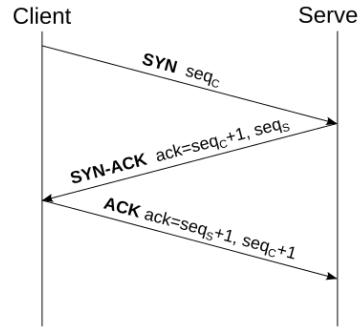
[1] In der Regel die Nummer des ersten Bytes der Payload, die im betreffenden Segment gesendet wird.

[2] In der Regel die Nummer des als nächstes erwarteten Bytes.

TCP-Handshake

Verbindungsaufbau (Initialisierung Buchhaltung)

- Client sendet SYN: seq_c
- Server antwortet SYN: seq_S, ACK: $seq_c + 1$
- Client bestätigt ACK: $seq_S + 1$



Client wie Server sind nun im bestätigten Besitz der Sequenznummer seq_S bzw. seq_C des Gegenübers.

Die Verbindung ist aufgebaut (**established**[4]).

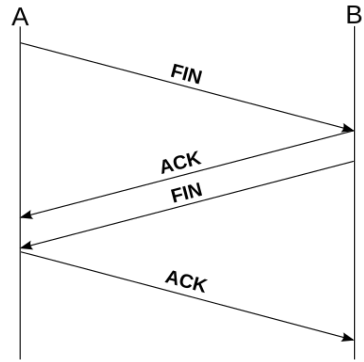
[4] `ss -t` zeigt TCP-Verbindungen in diesem Zustand an

TCP-Teardown

Verbindungsabbau (Ressourcenfreigabe)

- Teilnehmer sendet FIN
- Gegenüber bestätigt mit ACK ...
- ... und sendet ebenfalls FIN
- Letzte Bestätigung nochmal mit ACK

Die Verbindung ist abgebaut.



netcat – arbitrary TCP and UDP connections

Listening Server Socket[5]

```
netcat -l 1234   ##  -l listen,
                     -k when a connection is completed,
                                        listen for another one.
```

Verbindung zum Listening Socket[6]

```
netcat -4 localhost 1234
```

[5] Probe mit `ss -t -l -p`

[6] Kann natürlich auch von einem anderen Rechner aus erfolgen, Firewall deaktivieren!

Wireshark: TCP-Verbindung

Verbindungsaufbau:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	41350 → 1234 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
2	0.000015862	127.0.0.1	127.0.0.1	TCP	74	1234 → 41350 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=
3	0.000028300	127.0.0.1	127.0.0.1	TCP	66	41350 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSv

```
andi@flashgordon:~$ netcat -l 1234
```

```
andi@flashgordon:~$ netcat -4 localhost 1234
```

Wireshark: TCP-Verbindung

Verbindungsaufbau:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	41350 → 1234 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
2	0.000015862	127.0.0.1	127.0.0.1	TCP	74	1234 → 41350 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=
3	0.000028300	127.0.0.1	127.0.0.1	TCP	66	41350 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSv

```
andi@flashgordon:~$ netcat -l 1234
```

```
andi@flashgordon:~$ netcat -4 localhost 1234
```

Auf- und Abbau, ohne Datenübertragung:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	41354 → 1234 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
2	0.000016690	127.0.0.1	127.0.0.1	TCP	74	1234 → 41354 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=
3	0.000028523	127.0.0.1	127.0.0.1	TCP	66	41354 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSv
4	5.380679746	127.0.0.1	127.0.0.1	TCP	66	41354 → 1234 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=
5	5.380809693	127.0.0.1	127.0.0.1	TCP	66	1234 → 41354 [FIN, ACK] Seq=1 Ack=2 Win=65536 Len=
6	5.380835072	127.0.0.1	127.0.0.1	TCP	66	41354 → 1234 [ACK] Seq=2 Ack=2 Win=65536 Len=0 TSv

```
andi@flashgordon:~$ netcat -l 1234
andi@flashgordon:~$
```

```
andi@flashgordon:~$ netcat -4 localhost 1234
^C
andi@flashgordon:~$
```

Wireshark: TCP-Verbindung

Aufbau, Datenübertragung und Abbau Client-seitig:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	41350 → 1234 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
2	0.000015862	127.0.0.1	127.0.0.1	TCP	74	1234 → 41350 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=
3	0.000028300	127.0.0.1	127.0.0.1	TCP	66	41350 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSv
4	54.451298201	127.0.0.1	127.0.0.1	TCP	70	41350 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=
5	54.451336188	127.0.0.1	127.0.0.1	TCP	66	1234 → 41350 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSv
6	96.394535973	127.0.0.1	127.0.0.1	TCP	78	1234 → 41350 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=
7	96.394556671	127.0.0.1	127.0.0.1	TCP	66	41350 → 1234 [ACK] Seq=5 Ack=13 Win=65536 Len=0 TS
8	144.3854907…	127.0.0.1	127.0.0.1	TCP	94	41350 → 1234 [PSH, ACK] Seq=5 Ack=13 Win=65536 Len
9	144.3855154…	127.0.0.1	127.0.0.1	TCP	66	1234 → 41350 [ACK] Seq=13 Ack=33 Win=65536 Len=0 T
10	150.8576889…	127.0.0.1	127.0.0.1	TCP	66	41350 → 1234 [FIN, ACK] Seq=33 Ack=13 Win=65536 Le
11	150.8577943…	127.0.0.1	127.0.0.1	TCP	66	1234 → 41350 [FIN, ACK] Seq=13 Ack=34 Win=65536 Le
12	150.8578203…	127.0.0.1	127.0.0.1	TCP	66	41350 → 1234 [ACK] Seq=34 Ack=14 Win=65536 Len=0 T

```
andi@flashgordon:~$ netcat -l 1234
Hi!
Was gibt's?
Ach, passt schon, tschüss.
andi@flashgordon:~$
```

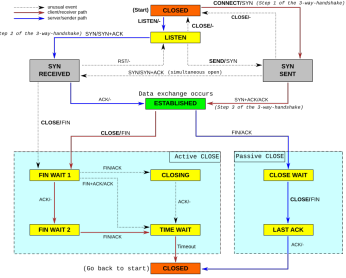
```
andi@flashgordon:~$ netcat -4 localhost 1234
Hi!
Was gibt's?
Ach, passt schon, tschüss.
^C
andi@flashgordon:~$
```

TCP bietet noch wesentlich mehr → Studium ;-)

Ausblick TCP

Auswahl einiger Abschnitte der TCP Wikipedia-Seite:

- Verbindungsaufbau und -abbau
  - Halb geschlossene Verbindungen
  - Halb offene Verbindungen
  - Verbindungsaufbau, Verbindungsabbau
  - Puffer
  - Drei-Wege-Handschlag
- Datenübertragung
  - TCP-/IP-Segment-Größe
  - Aufteilen der Anwendungsdaten auf TCP-/IP-Segmente
  - Retransmission Timer
  - Flusssteuerung
  - Überlaststeuerung/Staukontrolle (Congestion Control)
    - Algorithmus zur Überlaststeuerung
    - Slow Start und Congestion Avoidance
    - Fast-Retransmit und Fast-Recovery
    - Selective ACKs (SACK)
    - TCP-Tahoe und TCP-Reno
  - Überlaststeuerung als Forschungsfeld
- TCP-Prüfsumme und TCP-Pseudo-Header
- Datenintegrität und Zuverlässigkeit



[6] Sergiodc2, Marty Pauley, Scil100: https://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed_new.svg

Zusammenfassung und Rückblick

1. Wie werden Daten im Netz übertragen?
   - Layer 2: Ethernet Protokoll
   - Layer 3: Internet Protokoll
   - Layer 4: Prozesse verbinden
2. Das TFTProtocol
   - TFTP Anwendung
   - Der Zauberlehrling und weitere Defizite
   - RFCs: Requests for Comments
   - TFTP Option Extension
3. Das Transmission Control Protocol (TCP)
   - Buchhaltung
   - `netcat`: TCP-Verbindung
   - Ausblick TCP

Keep the sourcerer alive!

Zusammenfassung und Rückblick

1. Wie werden Daten im Netz übertragen?
   - Layer 2: Ethernet Protokoll
   - Layer 3: Internet Protokoll
   - Layer 4: Prozesse verbinden
2. Das TFTProtocol
   - TFTP Anwendung
   - Der Zauberlehrling und weitere Defizite
   - RFCs: Requests for Comments
   - TFTP Option Extension
3. Das Transmission Control Protocol (TCP)
   - Buchhaltung
   - `netcat`: TCP-Verbindung
   - Ausblick TCP

Keep the sourcerer alive!

Vielen Dank für Interesse und Aufmerksamkeit!