Ausnutzung verdeckter Kanäle am Beispiel eines Web-Servers

Diplomarbeit in Informatik

Abschlussvortrag

Hans-Georg Eßer

Lehr- und Forschungsgebiet Informatik IV
RWTH Aachen, 04.05.2005

Thema & Timing

Abstract: Modifizierter Apache-Server sendet Pakete ggf. zeitverzögert und übermittelt dadurch versteckte Nachrichten.

- Implementation (Sender und Empfänger)
- Kapazitätsbetrachtungen
- Fehlerkorrektur
- Einsatzmöglichkeiten

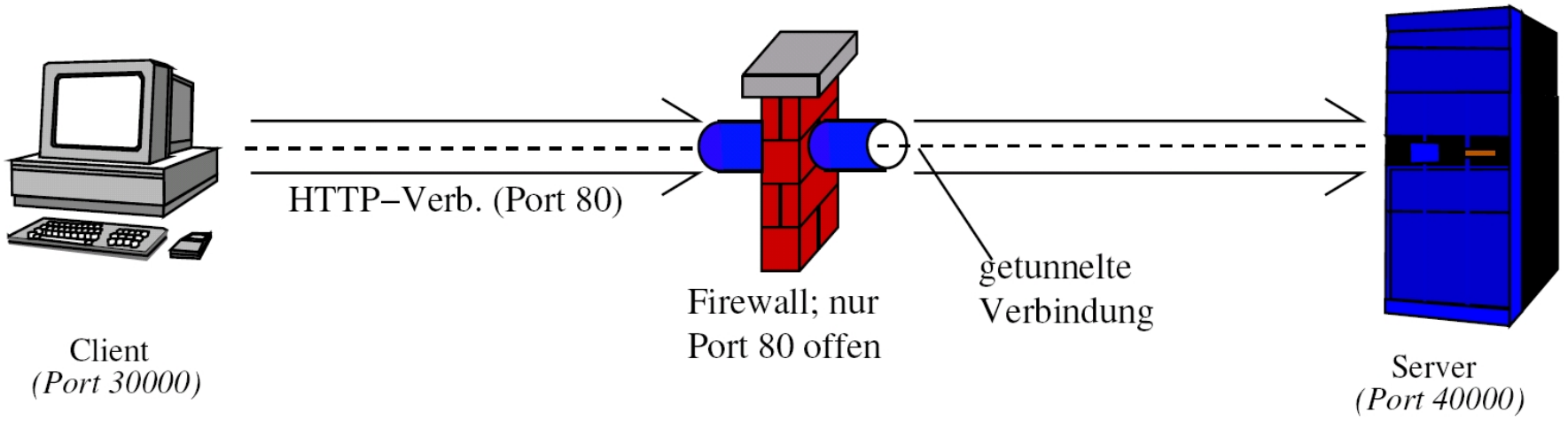
30 Min.	Vortrag
15 Min.	Demonstration
x Min.	Diskussion

Motivation (1/2): Trojanisches Pferd

- Trojaner: Programm mit unerwünschten Zusatz-„Funktionen“ (z. B. Einsatz für DDOS-Attacken)
- baut selbständig Netzwerkverbindungen auf, um mit seinem Besitzer zu kommunizieren
  (z. B. Information über neue IP-Adresse;
  Fern-Administration des infizierten PC)
- Problem für den Trojaner: Firewall mit restriktiver Policy (etwa: nur Port 80 offen)
- Lösung: Tunnel

Motivation (2/2): HTTP-Tunnel

Verbindung über HTTP tunneln
Firewall-Regeln umgehen



Unerwartete Nutzung des HTTP-Ports

Verdeckter Kanal (*covert channel*)

Butler W. Lampson

1973: „A note on the confinement problem“

- Kommunikationskanal zwischen zwei Prozessen
- Computersystem-Mechanismus wird in unerwarteter Weise genutzt, so dass Information zu einem nicht autorisierten Individuum fließt. [Amoroso 1994]
- Zwei Sorten: Zeit- und Ressourcenkanäle

Verdeckter Kanal – Ressourcenkanal

Ressourcenkanal

Gemeinsamer Zugriff auf eine Ressource, z. B.

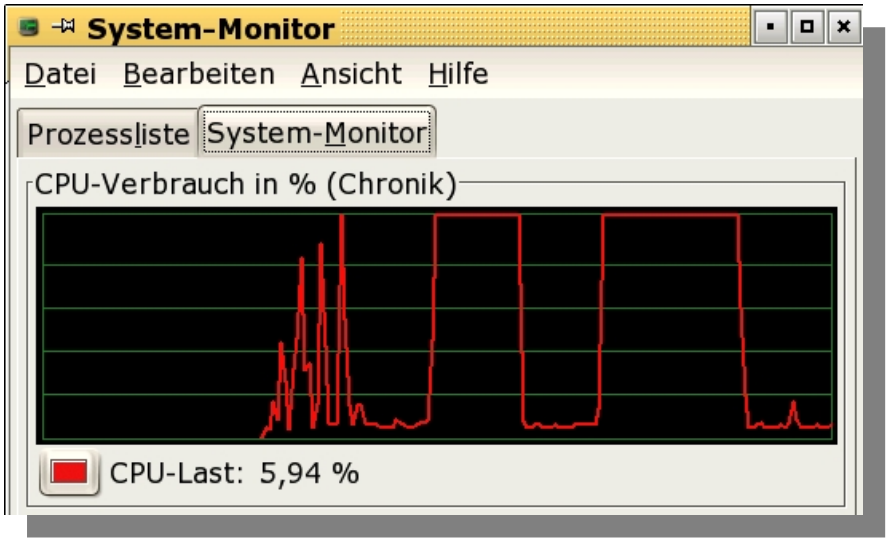
- Plattenplatz
- Prozessorzeit
- Hauptspeicher

```
aaa$ find / | sort > /dev/null
```

```
bbb$ cat /proc/loadavg
3.47 2.91 2.70 1/230
```

```
aaa$ sleep 300
```

```
bbb$ cat /proc/loadavg
0.00 0.00 0.00 1/230
```



Verdeckter Kanal – Zeitkanal

Zeitkanal

Sender beeinflusst zeitliches Verhalten, z. B.

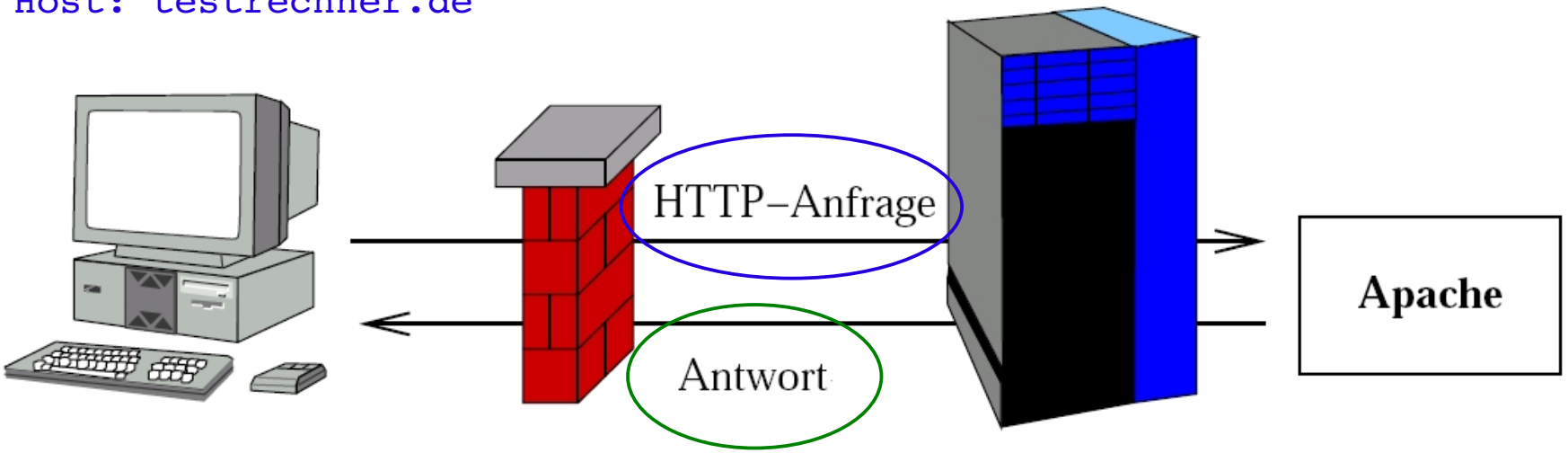
- Reaktionszeit eines Servers
- Zeit, bis Login-Dialog einen Fehler meldet
- Ausführungsdauer eines Kommandos

Empfänger beobachtet verändertes Verhalten und schließt daraus auf die Nachricht

Beteiligte Prozesse brauchen Zugriff auf eine „Uhr“

HTTP-Übertragung

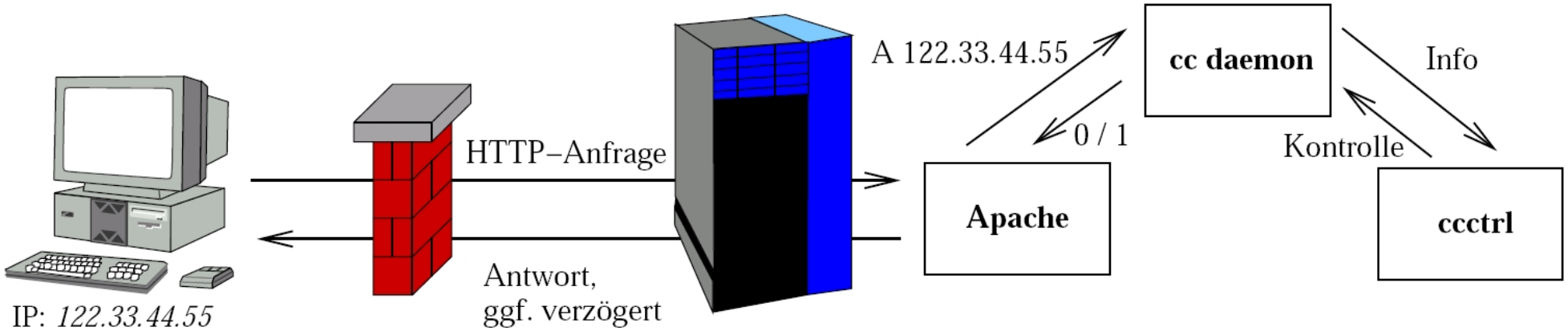
```
GET index.html HTTP/1.1
Host: testrechner.de
```




```
HTTP/1.1 200 OK
Date: Sun, 08 Aug 2004 17:07:16 GMT
Server: Apache/1.3.26 (Unix) Debian GNU/Linux PHP/4.1.2
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

ba2
<html>
...
```

Idee: HTTP-Päckchen verzögern



Apache unterteilt Datei in 4-KByte-Blöcke.
Für jeden Block:

- Apache fragt CC-Daemon:
  „Block an 122.33.44.55 verzögern?“
- CC-Daemon antwortet „1“ (ja) oder „0“ (nein)
- Falls „1“, verzögert Apache den Block

Implementation

Vier Haupt-Komponenten

Sender

a) Apache-Server
b) Kontroll-Daemon

Empfänger

c) HTTP-Proxy
d) Analyse-Tools

Implementation: Apache-Server (1/2)

- Apache 1.3.31 (*www.apache.org*)
- Funktionen verändern:
  `ap_send_fd_length()`
  `ap_send_mmap()`
- Variablen:
  `IOBUFSIZE`
  `MMAP_SEGMENT_SIZE`
- In der Schleife:

```
covert_ip = r->connection->remote_ip;
if (ap_covert_check_ip (covert_ip)) {
  sleepsec(AP_CC_SLEEPTIME);
}
```

Implementation: Apache-Server (2/2)

```
int ap_covert_check_ip (char *ip) {
  [...]
  /* "A " + IP-Adresse + Newline an Daemon senden */
  strcpy (daemon_request, "A ");
  strcat (daemon_request, ip);
  strcat (daemon_request, "\n");
  send (sd, daemon_request, strlen(daemon_request)+1, 0);

  /* Antwort empfangen und auswerten, Verbindung schließen */
  const int MAX_LEN=2;
  recv (sd, daemon_reply, MAX_LEN, 0);
  close (sd);
  int reply;
  if (daemon_reply[0] == '1')
    { reply = 1; } /* nur 1, wenn "1\n" empfangen wurde */
  else
    { reply = 0; }
  return reply;
}
```

Implementation: Kontroll-Daemon

- Daemon-Prozess `ccdaemon` (Python-Programm)
- akzeptiert Anfragen
  - vom Apache-Server und
  - von einem Kontrollprogramm:

  `A 192.168.1.1`
  = Paket verzögern? (Antwort „0“ oder „1“)

  `C msg 192.168.1.1 "Test Nachricht"`
  = Sende Nachricht „Test Nachricht“ an 192.168.1.1
- kodiert Anfragen in 0/1-Folgen und speichert diese in FIFO (für jede IP-Adresse eine)

Implementation: HTTP-Proxy (1/2)

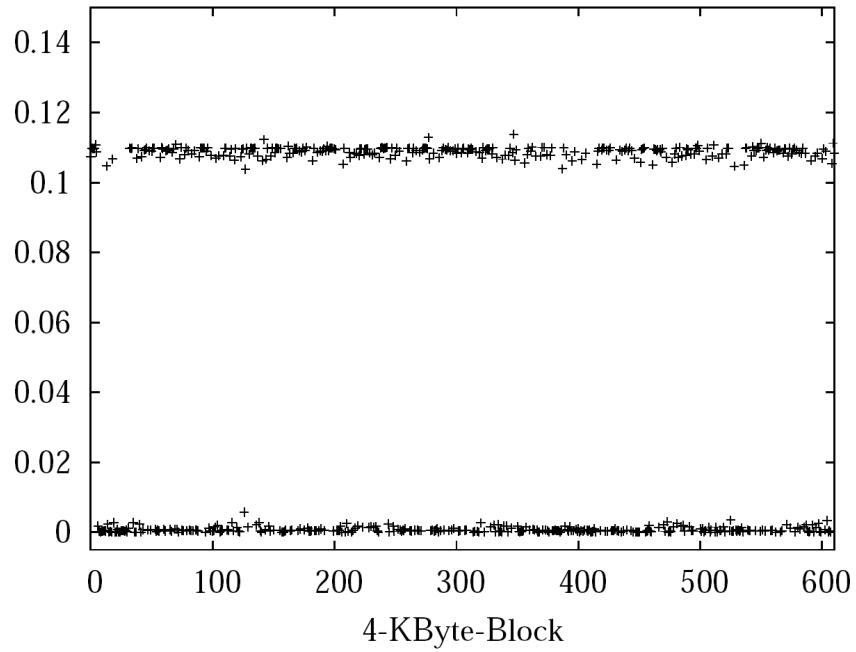
- Transparenter HTTP-Proxy `ccproxy` (Python-Programm)
- misst Verzögerungen zwischen
  - Anforderung eines 4-KByte-Blocks und
  - vollständigem Empfang dieses Blocks
- schreibt Messwerte in Log-Datei
- Vorsicht: *recv*(4096) empfängt evtl. < 4 KByte

Implementation: HTTP-Proxy (2/2)

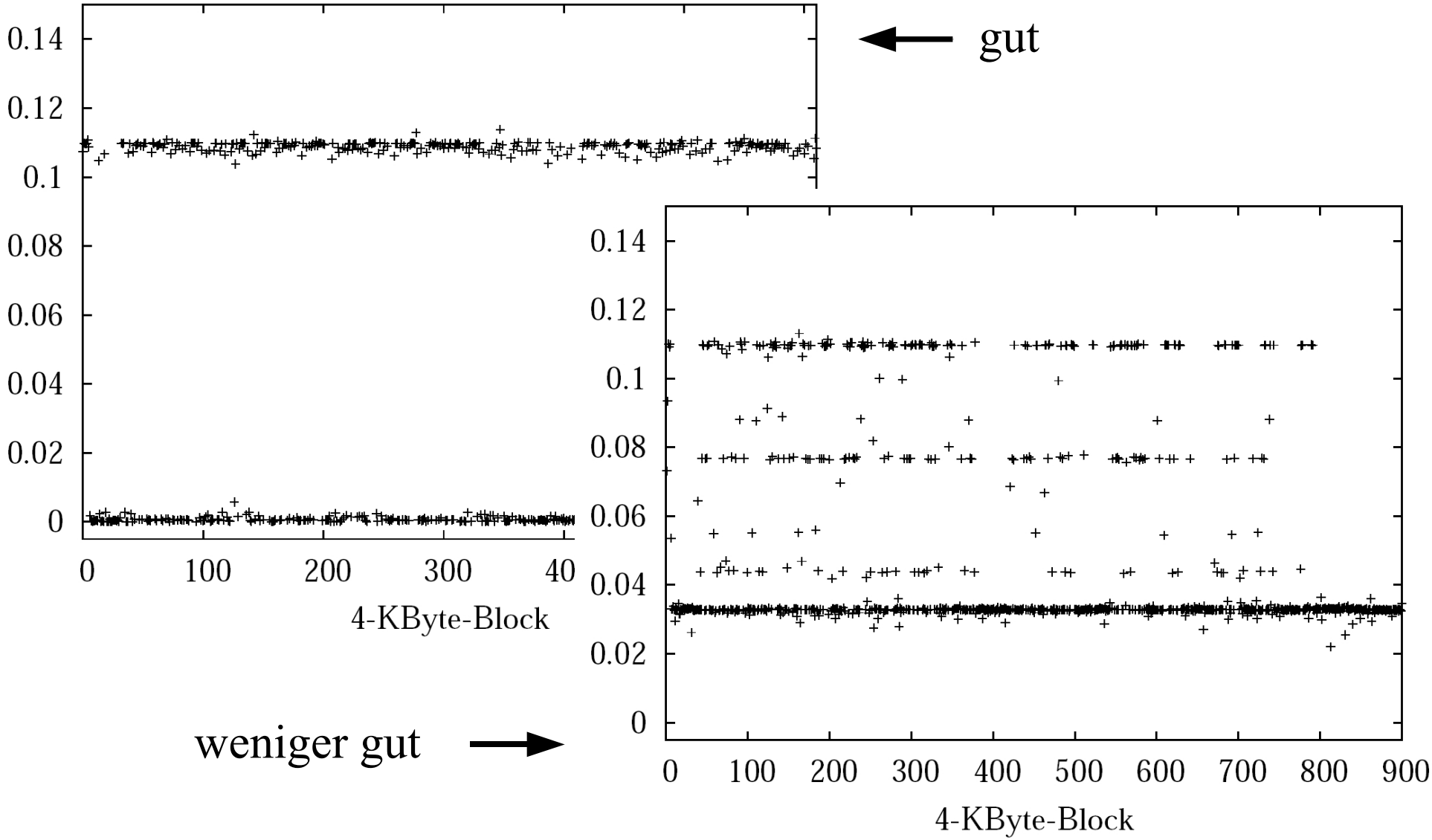
```
# Blockweise holen und zustellen
done=0
psize=4096              # Blockgröße für recv(): 4 KByte
times=[]; addedtime=0.0; recvsize=0
while not done:
  t0=time();   # vorher
  result=s.recv(psize-recvsize)
  t1=time();   # nachher
  recvsize += len(result)   # Wie viel Byte empf.?
  addedtime += (t1-t0)      # Verzögerung aufaddieren
  if recvsize >= psize:
    times.append(addedtime) # Verzögerung speichern
    recvsize -= psize;
    addedtime = 0.0
  c.send(result)
  if result=="": done=1
[...]
```

Implementation: Analyse-Tools (1/4)

- Messwerte betrachten
- Zwei „Häufungspunkte“
  - durchschnittliche unverzögerte Übertragungszeit
  - durchschnittliche verzögerte Übertragungszeit
- Interpretiere kurz und lang als 0 und 1
- Aber ...



Implementation: Analyse-Tools (2/4)



Implementation: Analyse-Tools (3/4)

Problem: Fragmentierung / Segmentierung

4-KByte-Blöcke nicht „in einem Rutsch“ übertragen

Untersuchung mit `strace` und `ethereal` zeigt:

- keine Aufteilung auf HTTP-Ebene (Apache verschickt 4-KByte-Blöcke, `strace`)
- MTU/MSS niedrig, um 1400-1500 (`ethereal`)
- TCP segmentiert die Pakete (`ethereal`)
- keine IP-Fragmentierung (da Größe passt); IP setzt DF

Implementation: Analyse-Tools (4/4)

Analyse-Programm `ccanalyse`

- teilt gemessene Zeiten in zwei Blöcke ober- und unterhalb Median (Gleichverteilung?)
- korrigiert Aufteilung: Werte eines Blocks, die näher am MW des anderen Blocks liegen, werden verschoben
- dann Dekodierung

Test-Szenarien; Kapazität

Test der verdeckten Kommunikation in drei Szenarien:

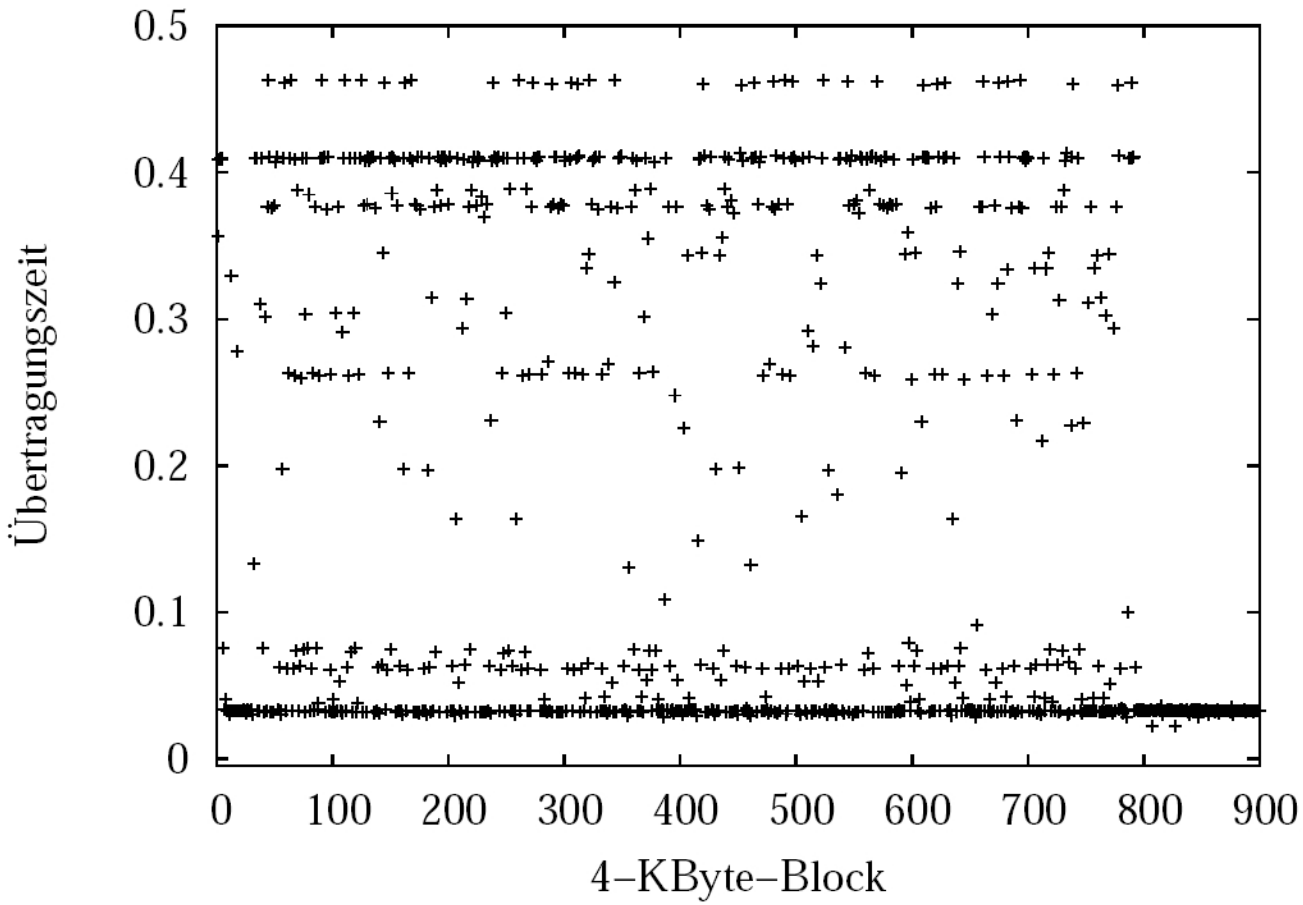
1. lokaler Test (nur ein PC)
2. Test im lokalen Netzwerk
3. Test im Internet (lastfrei / unter Last)

Relative Kapazität des verdeckten Kanals

- Maximal: 1 Bit / 4 KByte, also ca. 0,003 %
- Tatsächlich geringer durch Übertragungsfehler

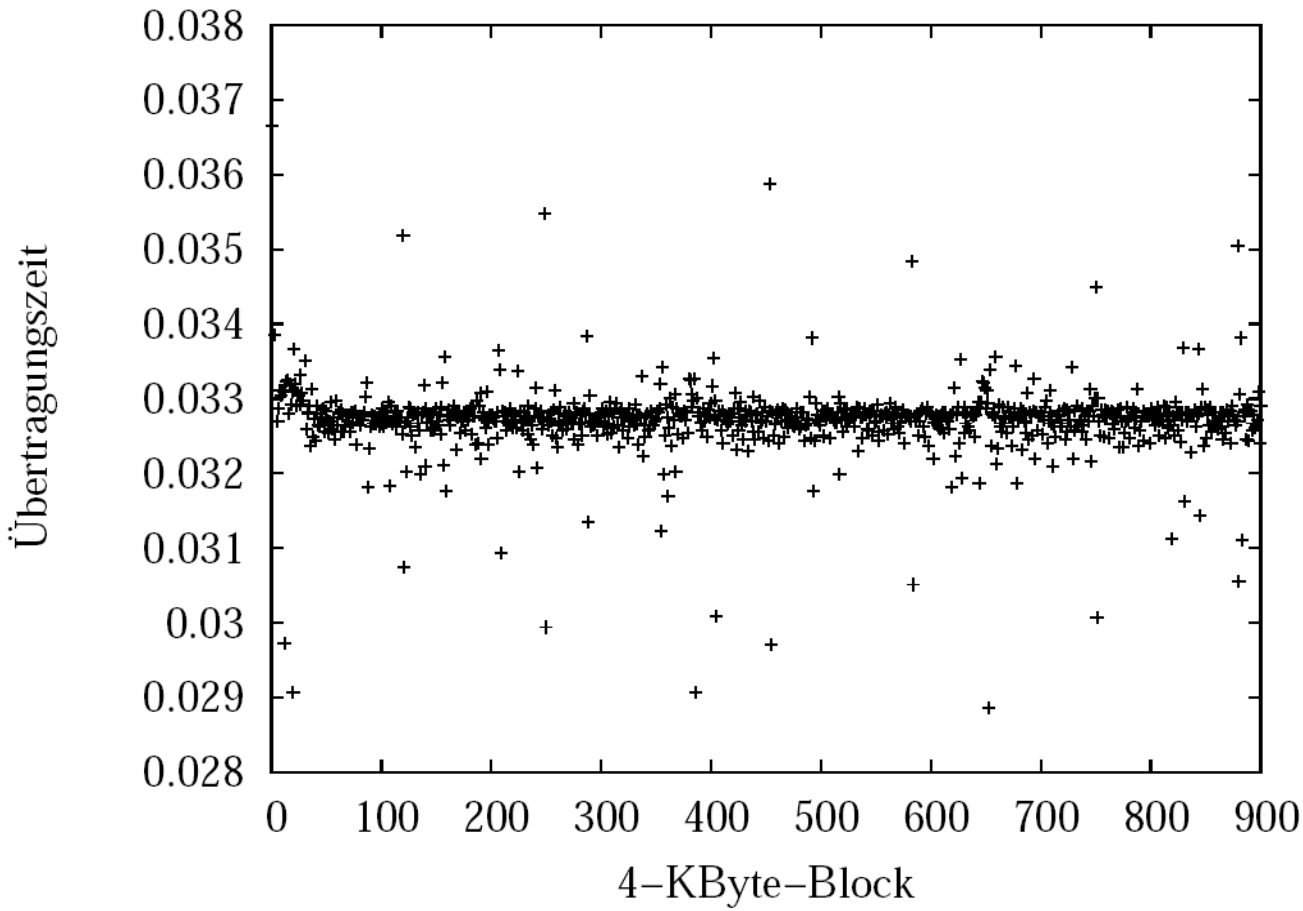
Beispiel 1: Internet mit $v = 0{,}4\,\mathrm{s}$

Internet, v = 0,4 s	
Gesamtwerte	
Median	0.0329700000
Mittelwert	0.1344422891
Abweichung	0.1570459346
Unterer Block	
Minimum	0.0219700000
Maximum	0.1325300000
Mittelwert	0.0361714355
Abweichung	0.0109453373
Oberer Block	
Minimum	0.0329800000
Maximum	0.6235500000
Mittelwert	0.3108148690
Abweichung	0.1423531659
Bitfehler	5
Fehlerrate	0,64 %

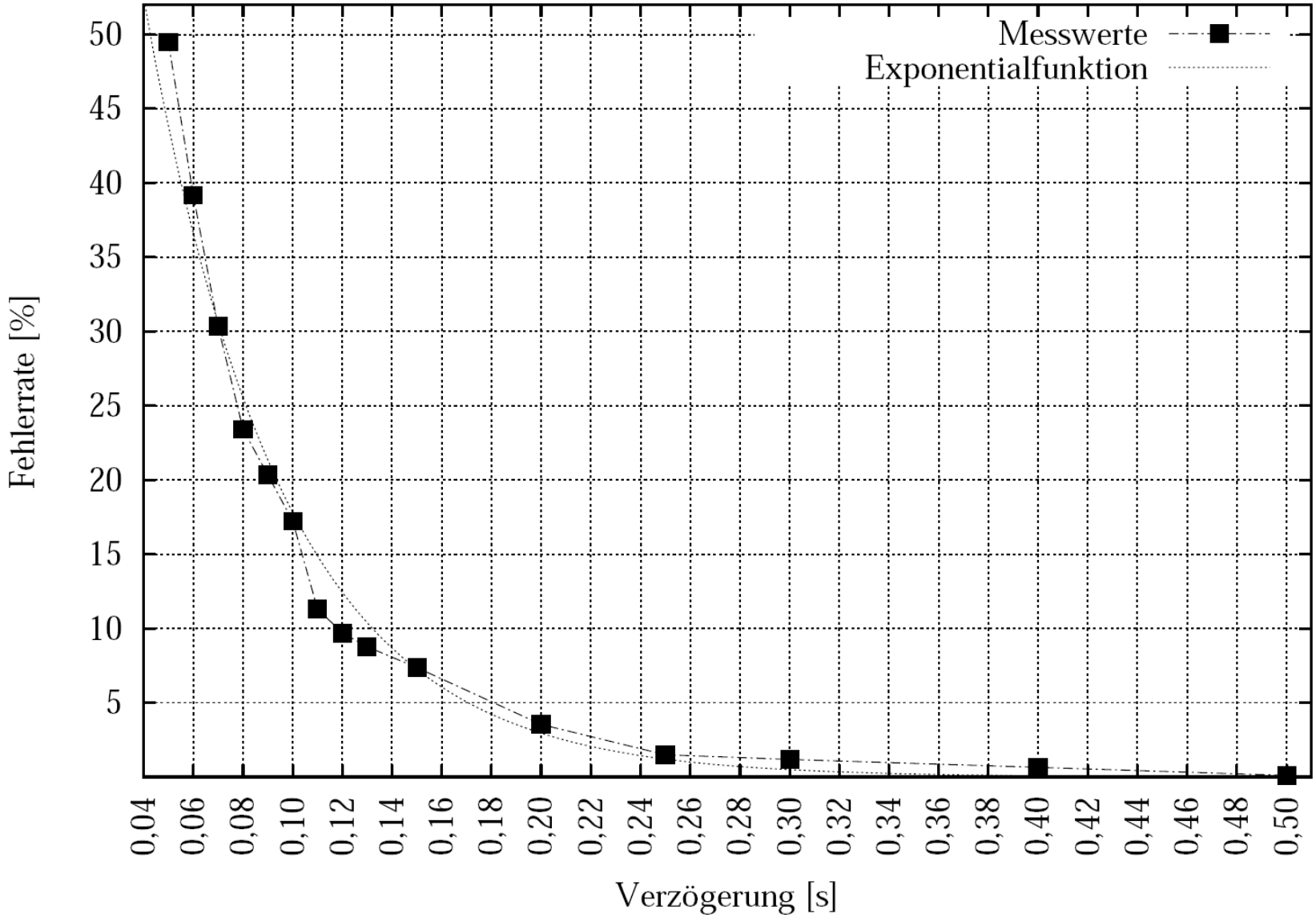


Beispiel 2: Internet mit $v = 0{,}05$ s

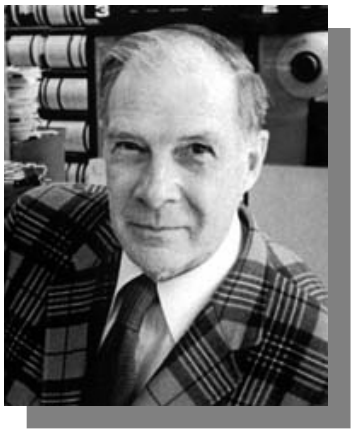
Internet, v = 0,05 s	
Gesamtwerte	
Median	0.0327600000
Mittelwert	0.0329584219
Abweichung	0.0069252043
Unterer Block	
Minimum	0.0273100000
Maximum	0.0329800000
Mittelwert	0.0326012435
Abweichung	0.0004558330
Oberer Block	
Minimum	0.0327700000
Maximum	0.2757000000
Mittelwert	0.0340166873
Abweichung	0.0137249371
Bitfehler	395
Fehlerrate	50,64 %



Fehlerraten (Internet)



Fehlerkorrektur (1/3)

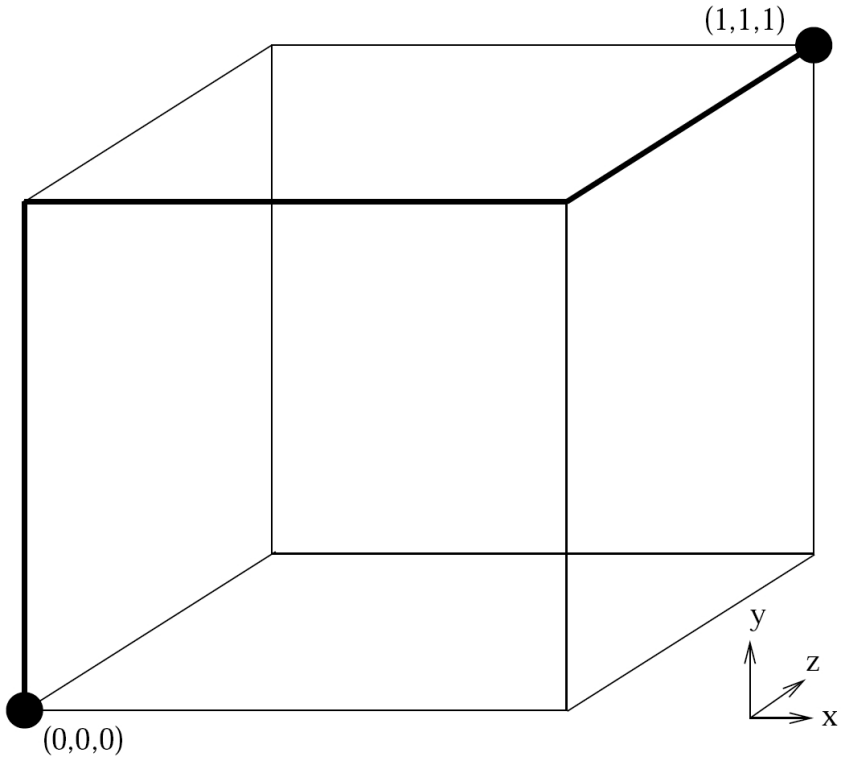


Richard W. Hamming

1950: „Error detecting and error correcting codes“

Hamming-Distanz

Code mit Hamming-Distanz $2n+1$ korrigiert n Bit-Fehler.

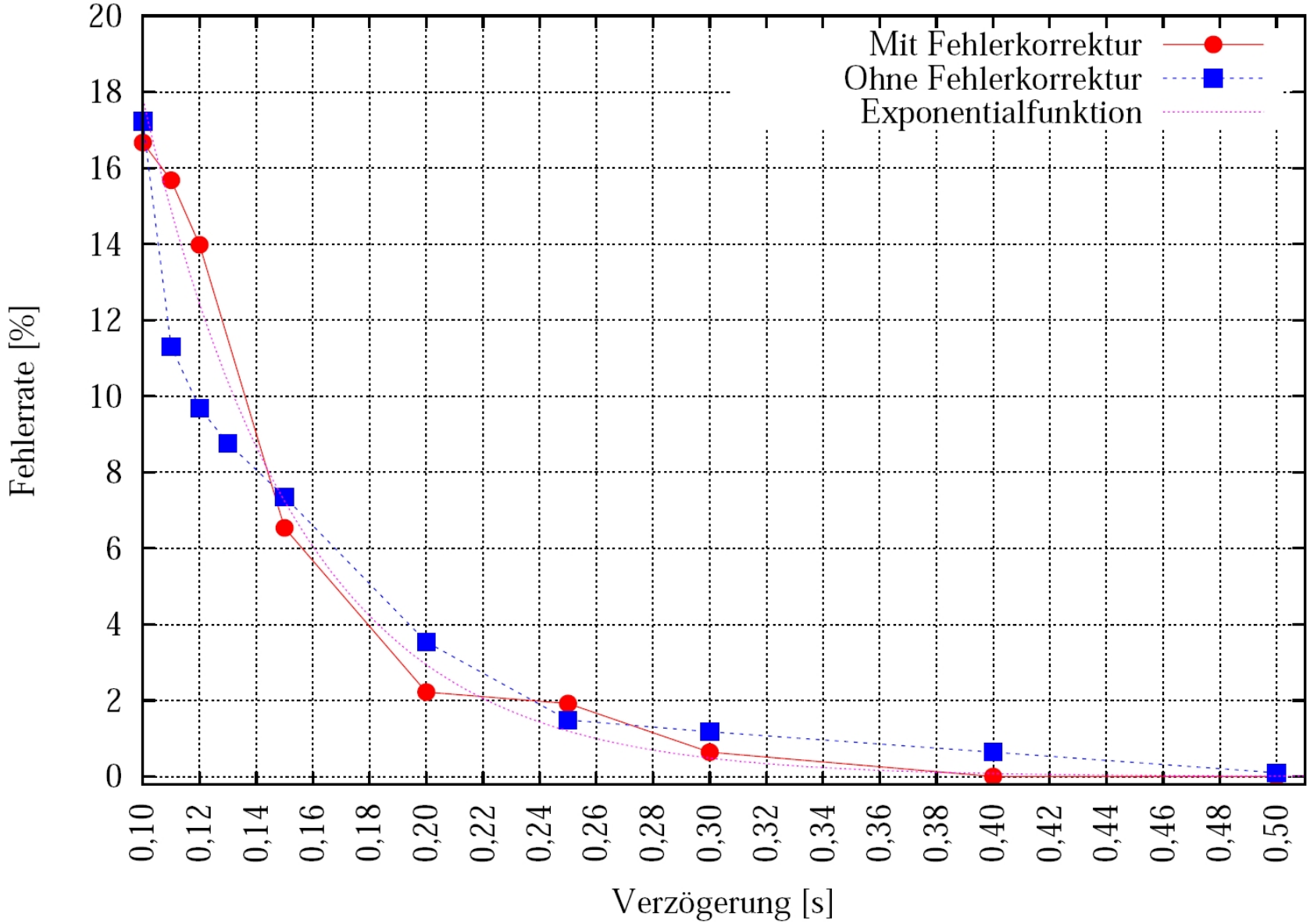


Fehlerkorrektur (2/3)

- Suche Code mit Hamming-Distanz d=3, um 1 Fehler zu korrigieren
- Eingabe: 6-Bit-Strings
- Kürzester Code mit d=3 besteht aus 10-Bit-Strings
- Generator-Matrix für den Code

$$y = xH, \text{ wobei } H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 1 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 1 & 0 & 0 & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 1 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}$$

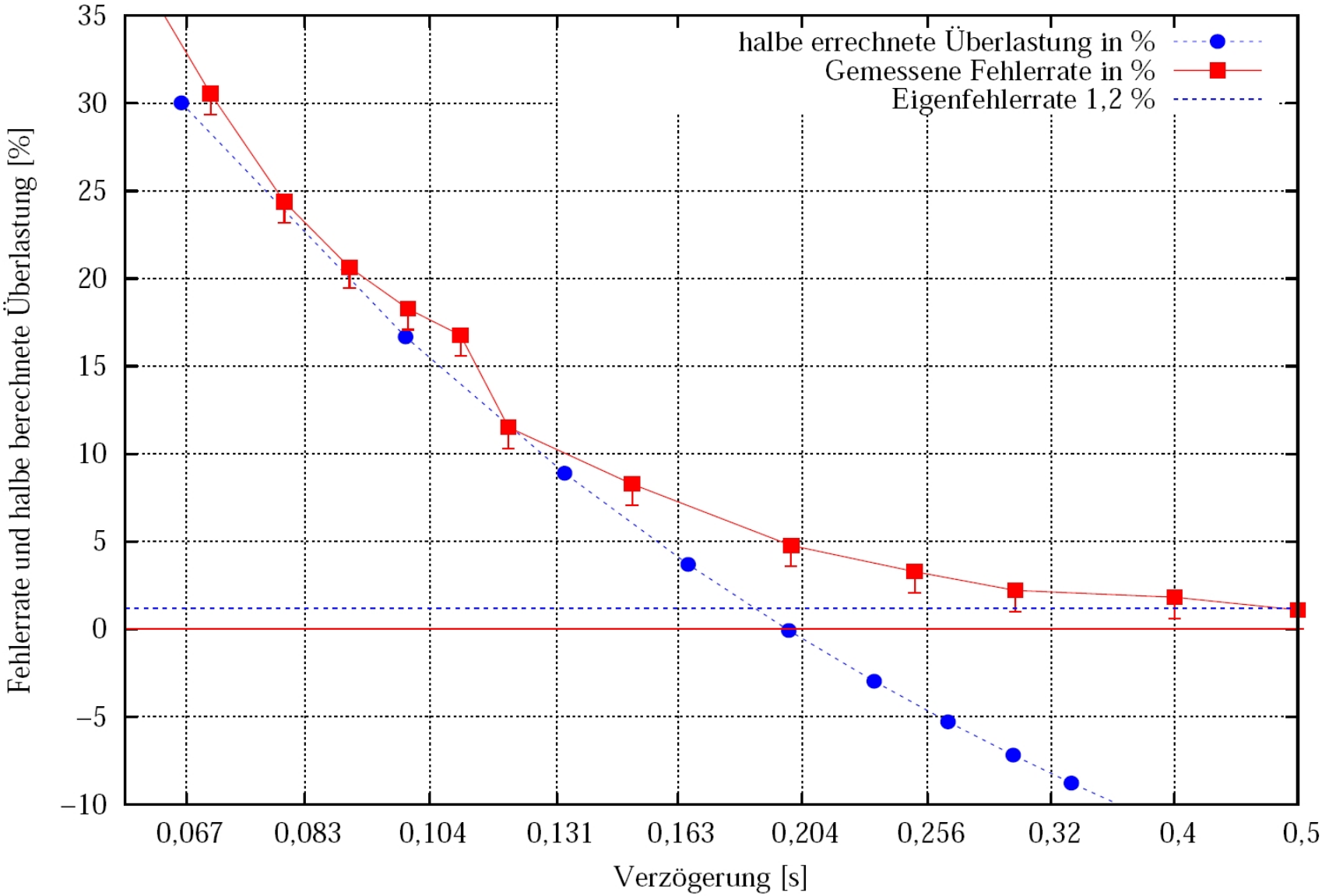
Fehlerkorrektur (3/3)



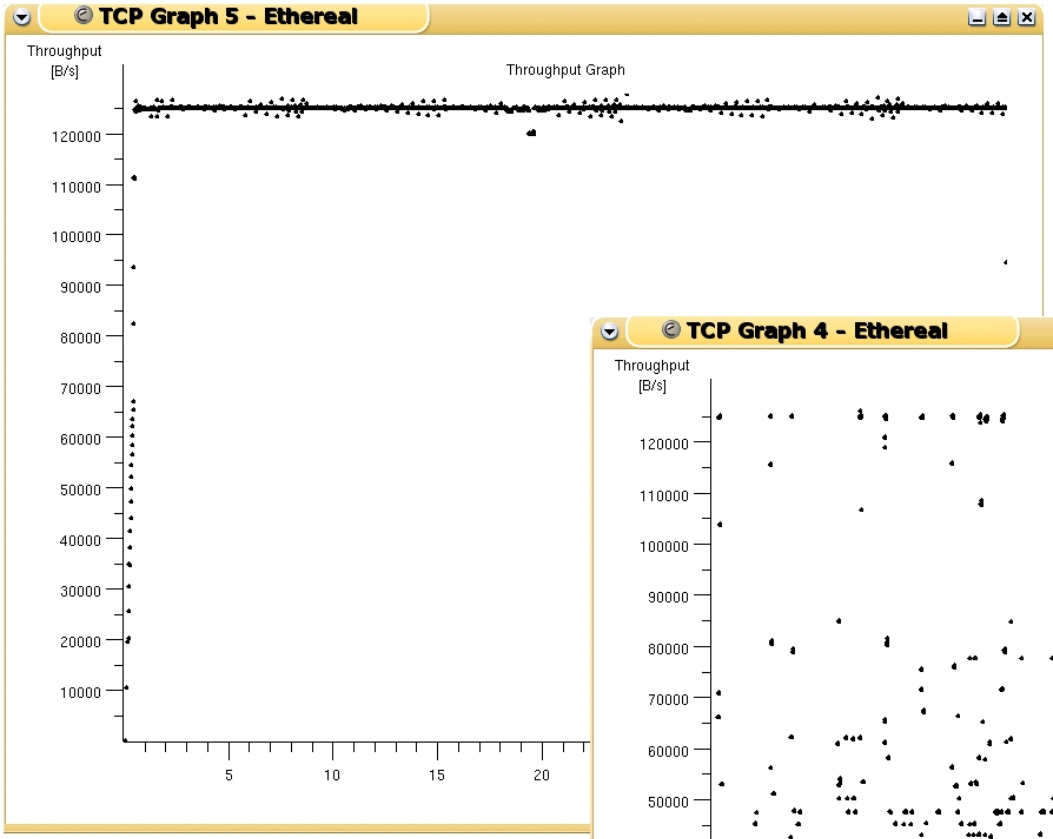
Kanalüberlastung vs. Fehlerrate (1/2)

- Theor. Berechnung der Kapazität:
  Kanal mit verschiedenen Übertragungszeiten
  (hier: 1/0, verzögert/unverzögert)
- Messung der Übertragungszeiten
- → Kanalüberlastung
- Vergleich mit gemessener Fehlerrate:
  Fehlerrate > ½ Kanalüberlastung

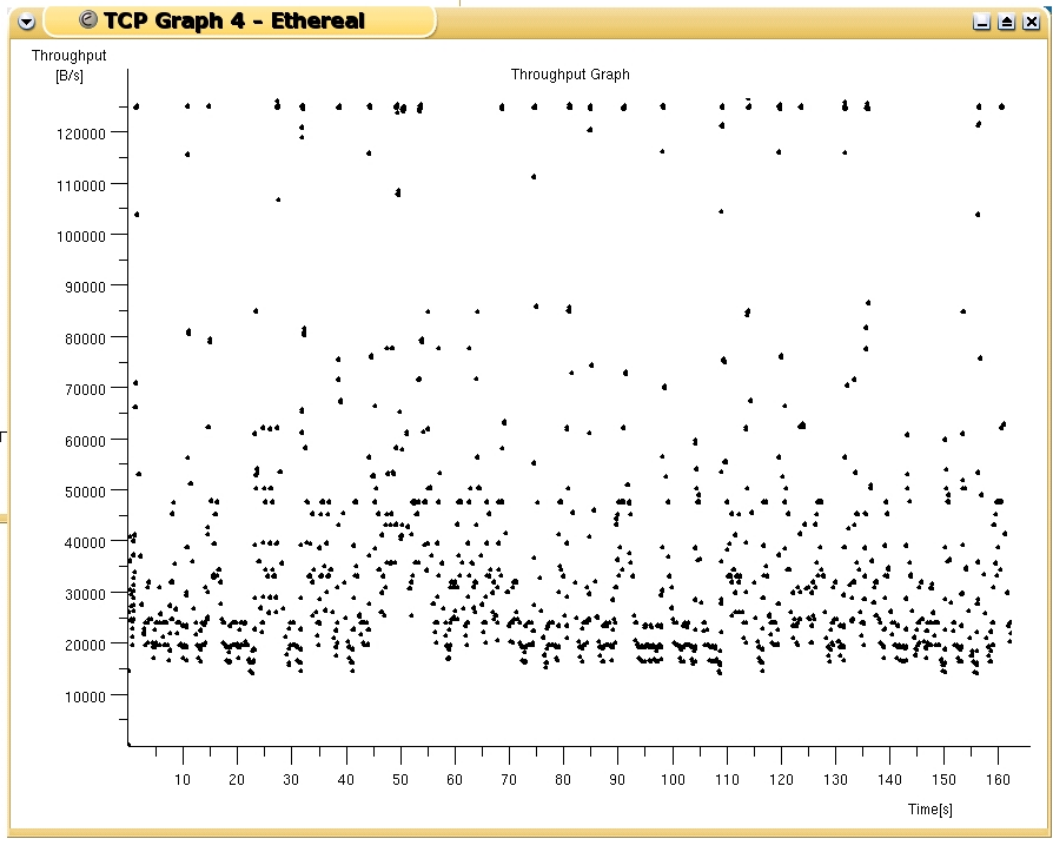
Kanalüberlastung vs. Fehlerrate (2/2)



Wie verdeckt ist der Kanal?



Datendurchsatz
(unverzögerte Übertragung)



Datendurchsatz
(verzögerte Übertragung)

Wozu kann man den Kanal verwenden?

- Kleines Datenvolumen in Relation zum Trägerkanal, daher nur eingeschränkter Nutzen
- Kurze Statusnachrichten
- Potentielle Sicherheitslücke (Was können *Andere* damit tun?)

Zusammenfassung

- Vier Komponenten entwickelt: Apache-Server (Patch), Daemon, HTTP-Proxy, Analyse-Tools
- Erfolgreiche Implementierung: Verdeckter Kanal durch zeitliche Veränderungen im HTTP-Transfer
- Wer Existenz *dieses* Kanals vermutet, findet ihn
- Hohe Fehlerraten, selbst bei größeren Verzögerungen
- Simple Fehlerkorrektur reduziert teilweise Fehlerrate (mehr Redundanz für weitere Verbesserung nötig)
- Vermutung: Fehlerrate hängt exponentiell von Verzögerungszeit ab (gestützt durch Messungen)

Ausblick / Erweiterungen

- Rückkanal
- Integration weiterer Verfahren: Verschlüsselung, Steganographie
- Interaktivität
- Mehrere Trägerdateien
- GUI
- TCP-Segmentierung

Andere Arbeiten

Typisch: Modifikation der Daten auf dem Trägerkanal

- MIX-Netze
- Infranet gegen Internet-Zensur
- LOKI2: Ping-Pakete

Es folgt eine Demonstration