

# **ITM-Praktikum**

## **Versuch 4: VoIP-/Videokonferenz-Systeme**

Andreas Klingler, Hannes Stahl, Simon Lüke

13. Januar 2010

# Inhaltsverzeichnis

<b>1</b>	<b>Vorbereitende Fragen</b>	<b>2</b>
1.1	MTU . . . . .	2
1.2	Daten im IP-Paket . . . . .	2
1.3	Paketverluste in TCP und UDP . . . . .	2
1.4	Sequence Number . . . . .	2
1.5	TCP-Verbindungsaufbau . . . . .	3
1.6	Beispiel zur Empfangsbestätigung . . . . .	3
1.7	Flusskontrolle . . . . .	3
1.8	Slow-Start-Algorithmus . . . . .	3
1.9	Maximierung der effektiven Datenrate . . . . .	4
<b>2</b>	<b>Versuchsdurchführung</b>	<b>4</b>
2.1	Fragmentierung von IP-Paketen . . . . .	4
2.2	Erzeugen/Messen von Netzwerktraffic . . . . .	4
2.2.1	Vorabmessung . . . . .	4
2.2.2	Variation der Windowsize . . . . .	4
2.3	Netzwerk-Emulation . . . . .	5
2.3.1	Richtung der Übertragung feststellen . . . . .	5
2.3.2	Datenrate bei Paket-Drop . . . . .	6
2.3.3	Datenrate bei Delay . . . . .	7
2.3.4	DSL 3000 Simulation . . . . .	9
2.3.5	DSL 1000 Simulation . . . . .	9

# 1 Vorbereitende Fragen

## 1.1 MTU

Die **Maximum Transfer Unit (MTU)** gibt die maximale Menge an Daten an, die über ein Paket in der Schicht 2 versendet werden können, sprich wie groß beispielsweise ein komplettes IP Frame sein darf. Die Größe dieses Datenbereichs ist von der verwendeten Technik/Hardware abhängig. Dazu ein paar Beispiele:

- Token Ring: 4464 Byte
- Ethernet: 1500 Byte
- Gigabit Ethernet: 9000 Byte
- PPPoE: 1492 Byte
- Wlan (802.11) 2312 Byte

Dies sind **nicht** die Framengrößen des ganzen Pakets auf Hardwareebene, sondern nur deren Datenbereich! Beim Beispiel Ethernet kommen z.B. noch 18 Byte Header hinzu, wodurch dieses Frame dann 1518 Byte auf Hardwareebene hat.

## 1.2 Daten im IP-Paket

Der fixe Headerbereich des IP Paketes umfasst 20 Byte, hinzu können bis zu 40 Byte optionale Informationen kommen. Dies muss alles in den Datenbereich des Ethernetframes hineinpassen. Somit verbleibt im IP-Frame zwischen 1480 und 1440 Byte für den Datenbereich übrig. Wenn mehr übertragen werden soll, müssen die Informationen fragmentiert und auf mehrere Ethernetframes verteilt und verschickt werden. Diese Fragmentierung kann im IP Layer für bis zu 64kB vorgenommen werden, alle Datenmengen größer 64kB muss TCP segmentieren (fragmentieren ≠ segmentieren!). IP-Fragmentierung kurz zusammengefasst:

1. das Flag 'more fragments' wird im IP Header gesetzt ⇒ zu dem Paket gehören weitere Fragmente,
2. zusammengehörende Fragmente eines Pakets bekommen im Header die gleiche Identification Nummer,
3. um die IP-Pakete wieder richtig aneinanderreihen zu können, wird im Headerfeld 'Offset' der Beginn des aktuellen Fragments in Bytes (!) relativ zum 0. Byte des Pakets angegeben (Bsp. im 1. Paket sind 1480 Byte Daten versendet worden, so steht im Offsetfeld des 2. Paketes 1480)

## 1.3 Paketverluste in TCP und UDP

Das **Transmission Control Protocol (TCP)** ist ein bidirektionales Protokoll, welches Pakete vermitteln und absichern kann, so dass sämtliche Pakete vollständig beim Empfänger eingegangen sind. Um das sicherzustellen, fallen natürlich mehr Headerdaten und Traffic an, woraus längere Übertragungszeiten und höhere Netzlast resultieren. Das **User Datagram Protocol (UDP)** dagegen ist auf Geschwindigkeit und Leichtgewicht konzipiert. So enthält es nur die wichtigsten Headerdaten um das Paket zuzustellen, jedoch ohne Kontrolleinrichtungen. Es ist damit ein 'verbindungsloses' Protokoll. Es kann z.B. sehr gut für Sprach- oder Videodaten verwendet werden, wo wenige fehlende Pakete gar nicht ins Gewicht fallen.

## 1.4 Sequence Number

Die **Sequence Number** nummeriert die TCP Pakete der Reihe nach, damit Sender und Empfänger wissen wie die segmentierten Pakete aneinander gehören. Nummeriert wird aufsteigend nach An-

zahl der versendeten Bytes. Wird ein leeres Paket gesendet, wird auch nicht hochgezählt; eine Ausnahme hiervon gibt es bei gesetztem SYN Flag: beim Handshake z.B. ist das SYN Flag gesetzt, dabei wird die Sequence Number trotz leerem Datenbereich um 1 erhöht. Die **Initial Sequence Number** wird zu Beginn der Verbindung zufällig generiert und wird als erste Sequence Number beim Handshake eingesetzt. Damit der Empfänger eine Bestätigung für den Paketeingang beim Server hat, legt er eine **Acknowledgement Number** fest. Diese ist nur gültig wenn das ACK-Flag gesetzt ist und ist die Sequence Number des nächsten erwarteten Paketes.

## 1.5 TCP-Verbindungsaufbau

Die Verbindung wird mit einem sogen. **3-Wege-Handshake** aufgebaut:

1. Der Client generiert die ISN=100 und setzt SYN=true ACK=false ACK-Number=egal, Paket wird versendet.
2. Der Server verwirft (RST) oder empfängt das Paket. Nach Empfang generiert er sich die ISN=5000 und setzt SYN=true, ACK=true, ACK-Number=101.
3. Der Client bestätigt den Verbindungsaufbau mit ACK=true, SEQ-Number=101, ACK-Number=5001.
4. ab nun können Daten versendet werden. Es ist nicht mehr eindeutig, wer Server/Client ist.

## 1.6 Beispiel zur Empfangsbestätigung

Die gegebene ACK-Nachricht bestätigt den Empfang des Paketes 530153981. Das nächste beim Empfänger eintreffende Paket sollte die SEQ-Number 530153982 und die ACK-Number 2721058443 haben (da keine Daten im Paket enthalten waren). Der Empfänger erwartet für die nächsten 63712 Byte Nutzdaten vom Sender kein Acknowledge mehr und hat dementsprechend mindestens 63712 Byte Pufferspeicher frei.

## 1.7 Flusskontrolle

Mittels der **Flusskontrolle** wird überwacht, dass der Empfänger die Datenpakete auch verarbeiten kann, sprich ob der Pufferspeicher des Empfängers noch Platz für weitere Pakete hat. Der Puffer kann voll werden, wenn der Anwendung für welche die Daten bestimmt ist, nicht schnell genug die Daten aus dem Puffer abholen/verarbeiten kann. Um den Puffer nicht überlaufen zu lassen, wird deshalb die **'Window Size'** verkleinert und kann sogar gleich Null werden, wenn der Puffer des Empfängers voll ist. Null bedeutet: 'Bitte Keine Pakete mehr'.

## 1.8 Slow-Start-Algorithmus

Beim **Slow-Start-Algorithmus** wird zu Beginn der Übertragung mit einer kleinen Datenrate begonnen, um die Kapazität des Netzes auszutesten. Sobald ein ACK-Paket vom Server eintrifft, wird das Fenster um eine MSS (Maximum Segment Size) erhöht, sprich es werden in der zweiten Runde doppelt so viele Pakete auf den Weg geschickt. Daraufhin versendet der Server in der Slow-Start-Phase auch zwei ACK-Pakete, der Client verdoppelt die MSS Zahl wiederum. Somit wächst die Datenrate exponentiell bis der Slow-Start-Threshold (Slow-Start-Schwelle) erreicht ist. Danach wird nur noch um jeweils ein MSS erhöht, bis die gewünschte Windowsize erreicht ist. *Bemerkung: Es ist zu beachten, dass eine MSS nicht nur ein Paket sein kann!*

## 1.9 Maximierung der effektiven Datenrate

- Durch Vergrößerung der Pakete auf Hardwareebene (Ethernetebene).
- Durch hohe Congestion Window Größen, wenn es die Netzlast erlaubt.
- Durch große Window-Sizes (weniger ACK-Nachrichten).

## 2 Versuchsdurchführung

### 2.1 Fragmentierung von IP-Paketen

```
ping server -s 8000
```

Da die so gesendete Pakete zu groß für eine Ethernet-MTU (1500 Bytes, es bleiben also max. 1480 Byte für Daten im IP Frame) sind, müssen sie fragmentiert werden. In Wireshark tauchen also 6 Pakete auf, die folgendermaßen nummeriert sind:

```
1. Paket: ID: 43036, Fragment Offset= 0, More Fragments = true
2. Paket: ID: 43036, Fragment Offset=1480, More Fragments = true
3. Paket: ID: 43036, Fragment Offset=2960, More Fragments = true
4. Paket: ID: 43036, Fragment Offset=4440, More Fragments = true
5. Paket: ID: 43036, Fragment Offset=5920, More Fragments = true
6. Paket: ID: 43036, Fragment Offset=7400, More Fragments = false
```

Damit bestätigt sich das in Vorbereitungsfrage 2 geschriebene Procedere.

### 2.2 Erzeugen/Messen von Netzwerktraffic

#### 2.2.1 Vorabmessung

```
Iperf-Server Befehl: iperf -s
Iperf-Client Befehl: iperf -c server
```

Eine erste Messung mit iperf ergab einen Datendurchsatz von 94Mbit/s. Es wird sich also um ein 100Mbit/s Netzwerk handeln. Dieser Wert ist recht gut, da die theoretischen 100Mbit/s kaum erreichbar sind. Mögliche Ursachen könnten sein: Protokolloverhead und Hardwareinkompatibilität.

#### 2.2.2 Variation der Windowsize

Mit dem Befehl

```
iperf -c server -w <Window-Groesse>
```

erhält man, unter Variation der <Window-Groesse>, die Messreihe in Tabelle 2 mit dem dazugehörigen Schaubild aus Abbildung 1.

	Hallo	Du	Da		
Windowsize [kByte]	2	4	8	16	32
	44,8	86,5	93,4	93,6	93,2

Tabelle 2: Datenrate über Window-Size

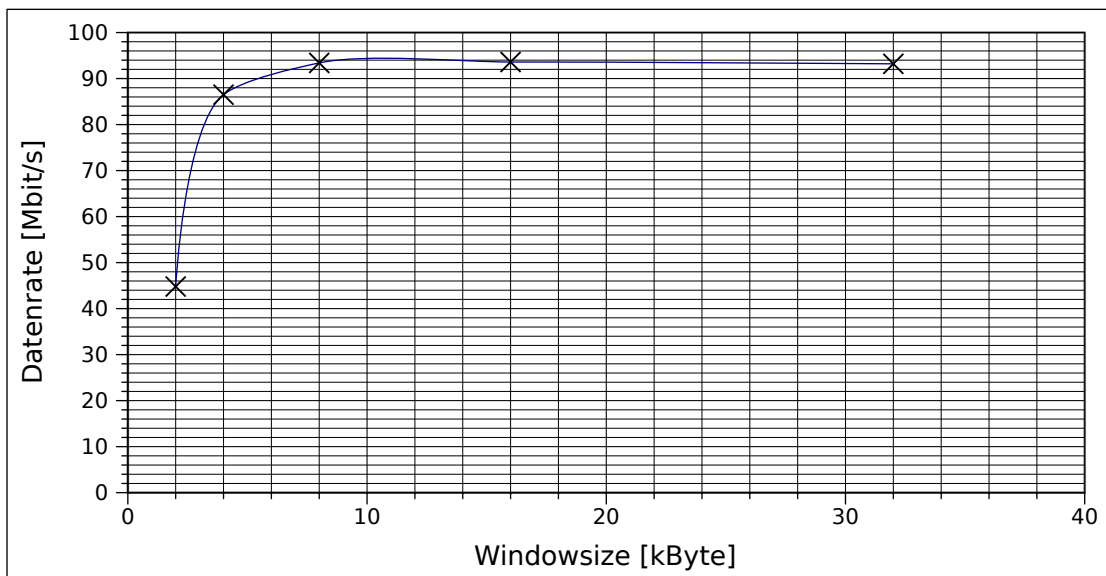


Abbildung 1: Datenrate über Window-Size

Es wird deutlich, dass sich bei Windowgrößen zwischen 4k und 8k die Übertragungsraten akzeptabel an die 94Mbit/s annähern.

## 2.3 Netzwerk-Emulation

### 2.3.1 Richtung der Übertragung feststellen

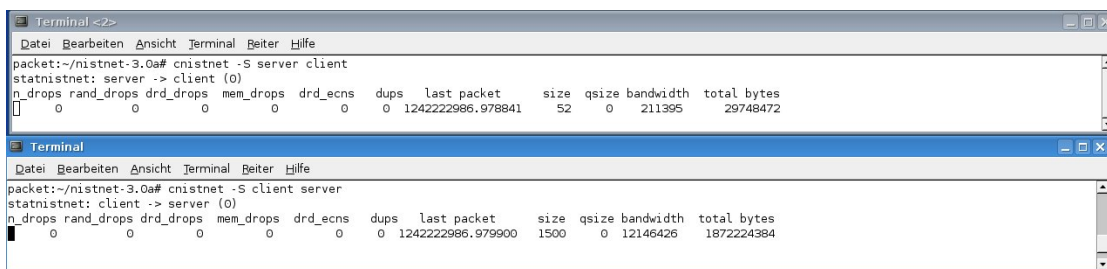


Abbildung 2: Screenshot Bidirektionale Emulation

Wie man an dem Screenshot sehen kann, ist unter der Spalte 'Size' die Zahl in der Messung

Client → Server 1500 Byte groß, was einer MTU entspricht.

⇒ Die großen Daten fließen vom Client zum Server.

### 2.3.2 Datenrate bei Paket-Drop

Paket-Drop Rate [%]	0	3	6	9	12	15
Datenrate [MBit/s]	93,7	42,0	10,8	4,4	2,0	1,5

Tabelle 3: Datenrate über Paket-Drops

No.	Time	Source	Destination	Protocol	Info
416	0.855554	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [ACK] Seq=272089 Ack=1 Win=5840 Len=1448 TSV=1046919 TSEr=1042882
417	0.855640	10.1.1.1	10.1.1.1	TCP	5001 > 3325 [ACK] Seq=1 Ack=272089 Win=86880 Len=0 TSV=1042883 TSEr=1046919
418	0.855678	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [PSH, ACK] Seq=273537 Ack=1 Win=5840 Len=1448 TSV=1046919 TSEr=1042882
419	0.855802	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [ACK] Seq=274985 Ack=1 Win=5840 Len=1448 TSV=1046919 TSEr=1042882
420	0.855886	10.1.1.1	10.1.1.1	TCP	[TCP Dup ACK 417#1] 5001 > 3325 [ACK] Seq=1 Ack=272089 Win=86880 Len=0 TSV=1042883 TSEr=1046919 SLE
421	0.855925	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [ACK] Seq=276433 Ack=1 Win=5840 Len=1448 TSV=1046919 TSEr=1042882
422	0.856009	10.1.1.1	10.1.1.1	TCP	[TCP Dup ACK 417#2] 5001 > 3325 [ACK] Seq=1 Ack=272089 Win=86880 Len=0 TSV=1042883 TSEr=1046919 SLE
423	0.856048	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [ACK] Seq=277881 Ack=1 Win=5840 Len=1448 TSV=1046919 TSEr=1042883
424	0.856132	10.1.1.1	10.1.1.1	TCP	[TCP Dup ACK 417#3] 5001 > 3325 [ACK] Seq=1 Ack=272089 Win=86880 Len=0 TSV=1042883 TSEr=1046919 SLE
425	0.856171	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [PSH, ACK] Seq=279329 Ack=1 Win=5840 Len=1448 TSV=1046919 TSEr=1042883
426	0.856255	10.1.1.1	10.1.1.1	TCP	[TCP Dup ACK 417#4] 5001 > 3325 [ACK] Seq=1 Ack=272089 Win=86880 Len=0 TSV=1042883 TSEr=1046919 SLE
427	0.856310	10.1.1.1	10.1.1.1	TCP	[TCP Fast Retransmission] 3325 > 5001 [ACK] Seq=272089 Ack=1 Win=5840 Len=1448 TSV=1046920 TSEr=1042883
428	0.856379	10.1.1.1	10.1.1.1	TCP	[TCP Dup ACK 417#5] 5001 > 3325 [ACK] Seq=1 Ack=272089 Win=86880 Len=0 TSV=1042883 TSEr=1046919 SLE
429	0.856516	10.1.1.1	10.1.1.1	TCP	5001 > 3325 [ACK] Seq=1 Ack=280777 Win=84736 Len=0 TSV=1042883 TSEr=1046920
430	0.856702	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [ACK] Seq=280777 Ack=1 Win=5840 Len=1448 TSV=1046920 TSEr=1042883
431	0.856824	10.1.1.1	10.1.1.1	TCP	3325 > 5001 [ACK] Seq=282225 Ack=1 Win=5840 Len=1448 TSV=1046920 TSEr=1042883

Abbildung 3: Screenshot TCP-Datenverlust

Der Rechner 10.1.1.1 ist der Server, an den die Daten vom Client übermittelt werden sollen. Der Server selbst soll keine Daten versenden, sondern nur die empfangenen Daten bestätigen. Die Wiresharkaufzeichnung entstand am Packet-Rechner an der Client Karte (vor dem Drop!).

- Paket 416 hat die Sequenznummer 272089. Es kommt vom Client und soll zum Server - es wird jedoch von NIST gedroppt!
- Paket 417 ist ein Ack vom Server für 272088 empfangene Bytes. Der Server erwartet das Paket 272089
- Paket 418 sind Daten vom Client mit höherer Sequenznummer als 272089
- Paket 419 —
- Paket 420 bestätigt nochmals den Empfang von 272089 Bytes, da mit Paket 418 für den Server klar war, dass ein Paket fehlt
- Paket 421, 423, 425: der Client sendet weiterhin Daten mit hoher Sequenznummer
- Paket 422, 424, 426: der Server bestätigt auf jedes Datenpaket mit Ack=272089
- Paket 427: der Client versendet das verlorene Paket 272089 nochmals
- Paket 428: 69 ns nach Paket 427 trifft noch ein Ack für 272089. Dieses Paket resultiert vermutlich nur aus den Signallaufzeiten.
- Paket 429: Der Server hat Paket 272089 bekommen und bestätigt mit einem ACK Paket alle Daten bis Ende Paket 425

### 2.3.3 Datenrate bei Delay

Verzögerung Client → Server, Werte der Messreihe gemittelt über drei Messungen:

Delay [ms]	5	10	20	50	100	200	500
Datenrate [Mbit/s]	93,7	90,9	47,7	19,5	9,7	4,3	0,598

Tabelle 5: Messung - Verzögerung Client → Server

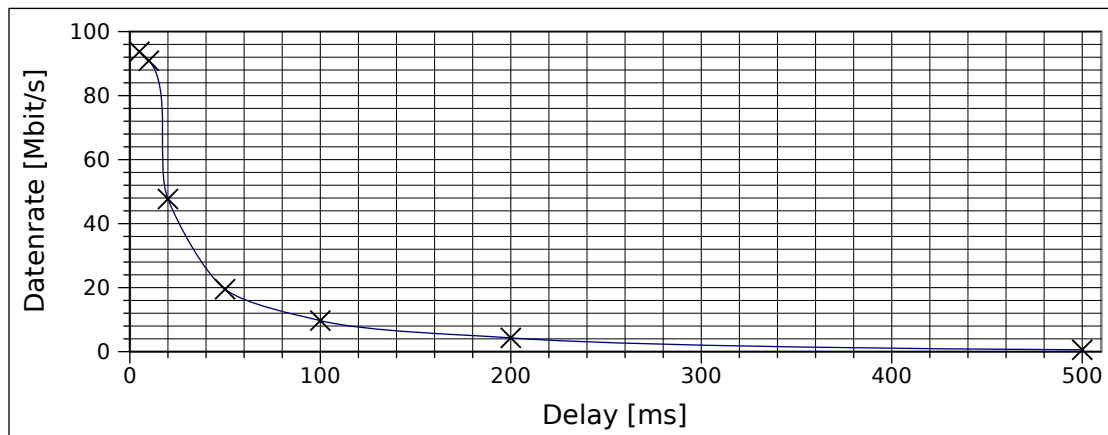


Abbildung 4: Datenrate über Delay (Verzögerung Client → Server)

Es ist einfach zu sehen, dass für das Einbrechen der Übertragungsrate die Verzögerung vom Client zum Server verantwortlich ist: die Übertragungsrate in Messung 7 bricht im Gegensatz zu den Messungen 5 und 9 nicht so plötzlich ein. Es fällt also nicht so sehr ins Gewicht, werden die „sporadischen“ ACK-Pakete vom Server verzögert werden, dies führt nur schwach zu einer Verminderung der Übertragungsrate; die Last entsteht durch versenden von „Nutzdaten“ vom Client zum Server.



Verzögerung Server  $\rightarrow$  Client, Werte der Messreihe gemittelt über drei Messungen:

Delay [ms]	5	10	20	50	100	200	500
Datenrate [Mbit/s]	1,59	1,58	1,56	1,41	1,28	1,05	0,591

Tabelle 7: Messung - Verzögerung Server  $\rightarrow$  Client

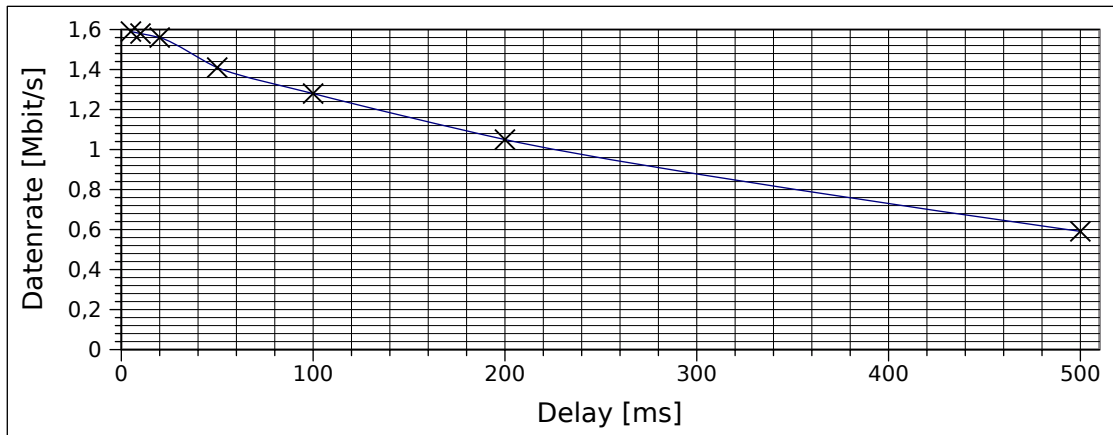


Abbildung 5: Datenrate über Delay (Verzögerung Server  $\rightarrow$  Client)

Beidseitige Verzögerung: Server  $\leftrightarrow$  Client, Werte der Messreihe gemittelt über drei Messungen:

Delay [ms]	5	10	20	50	100	200	500
Datenrate [Mbit/s]	90,8	48,1	24,4	9,8	4,7	2,1	1,3

Tabelle 9: Messung - Verzögerung Server  $\leftrightarrow$  Client

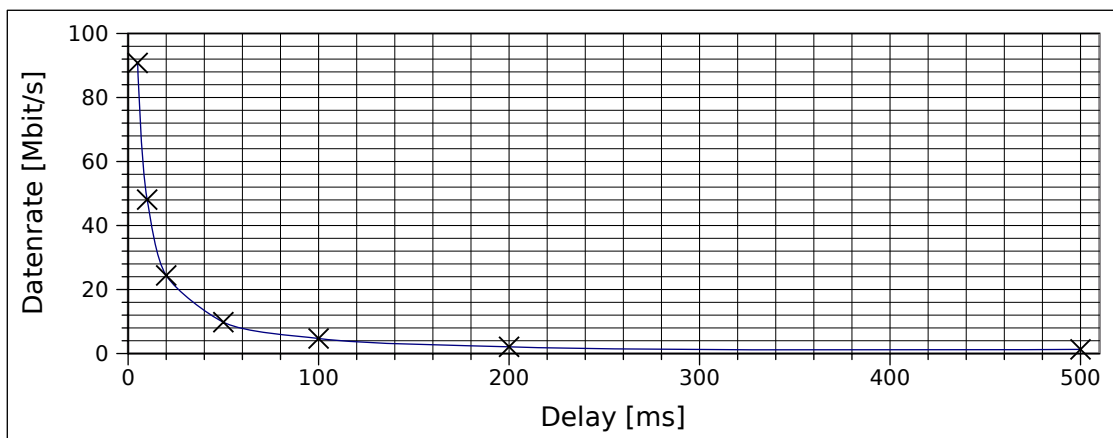


Abbildung 6: Datenrate über Delay (Verzögerung Server  $\rightarrow$  Client)

### 2.3.4 DSL 3000 Simulation

1. Bandbreite bei DSL 3000:

	Bandbreite [kBit/s]	Bandbreite [kByte/s]
Upstream	3072	384
Downstream	384	48

NistNet erwartet die Bandbreiteangaben in kByte/s, also müssen die Werte aus der dritten Spalte eingetragen werden.

Wir haben als Delay 20 ms eingestellt, was leider bei DSL-Leitungen ohne Fastpath unrealistisch ist. Eine übliche DSL Leitung weist ca. die doppelte Ping-Zeit auf (ca. 40ms).

2. **optimale Windowsize (wenig Traffic):** Bei kaum ausgelasteter Verbindung haben wir empirisch die optimalen Windowgrößen bestimmt. Es zeichnet sich in Downloadrichtung ein starker Knick bei 1500 Byte ab, während bei Upstreamdaten der Knick bei 7500 Byte lag. Die Windowsizes sollten höher als diese Knick-Größe liegen, da sonst hohe Datenrateneinbusen hinzunehmen sind.
3. **Pingzeiten bei voll ausgelasteter Verbindung:**
  - a) Downstream voll ausgelastet: Pingzeit von 300-400ms
  - b) Upstream voll ausgelastet: Pingzeit von 2,5-3s
  - c) Up- und Downstream voll ausgelastet: Pingzeit von 3-3,6s

### 2.3.5 DSL 1000 Simulation

Die Windowsize hat sich nicht verändert, wie aus einer Wireshark Messung schnell ersichtlich war. Nach Absprache mit dem Versuchleiter wurde vermutet, dass es sich evtl. um ein Kernelproblem handelt.