– RDUe Labor –
Rechnernetze und Datenübertragung
Laborbericht über Versuch:
NIC Performancemessung

Andreas Hofmeier

Auftraggeber:	Prof. Dr.-Ing. Kai-Oliver Detken, Fachhochschule Bremen
Durchführung am:	19.03.2004, 23.04.2004 und 07.05.2004
Ort der Durchführung:	FH Bremen, Flughafenalle 10, Raum: I222
Abgabe am:	28.07.2004

Versuchsgruppe

Andreas Hofmeier
Axel Schmidt

Zusammenfassung

.
In diesem Versuch wurde die Performance zweier mit einem CAT-5 Cross-Over-Cable verbundenden NICs (Netzwerkkarten) gemessen. Also die Zeit in welcher eine gewisse Menge an Daten von Rechner A mit NIC A zu Rechner B mit NIC B übertragen wurde.

Die Messungen erfolgten mit einem speziellem Programm, welches direkt TCP und UDP-Pakete versendete. Um Vergleiche anstellen zu können wurden die Messungen mit verschiedenen Sende-/Empfangspuffern und unterschiedlichen Betriebssystemen durchgeführt.

Inhaltsverzeichnis

1 Versuchsziele

Mit diesem Versuch sollte geklärt werden in wie weit sich Änderungen des Sende-/Empfangspuffers und das verwendete Betriebssystem auf die Datenübertragungsrate auswirkt. Es sollte ein Performancetest der Netzwerkkarten (NICs) und des Netzwerkes an sich (in diesem Fall ein CAT-5 Cross-Over-Cable) vorgenommen werden. Die Messung der Performance der Anwenderprogramme bzw. der Protokolle dieser war nicht erwünscht. Des weiteren sollte ein Grundverständnis des Netzwerkstacks (TCP/UDP über Ethernet) erlangt werden.

2 Vorbetrachtungen

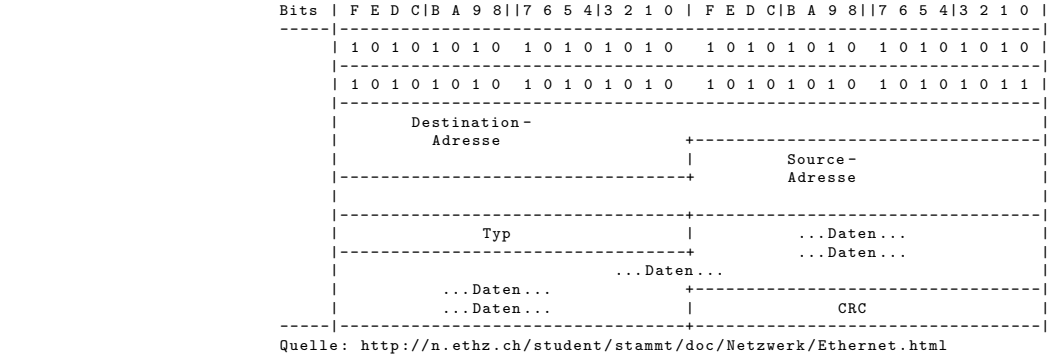
2.1 Schichten

Schicht	OSI-Schicht	In unserem Fall verwendet	
Anwendungsschicht	Application Layer Presentation Layer Session Layer	netperf	
Transportschicht	Transportschicht	TCP	UDP
Internetschicht	Netzwerkschicht	IP	
Netzwerkschicht	Sicherungsschicht Bitübertragungsschicht	Ethernet	

In diesem Abschnitt wurden teilweise Texte von `http://n.ethz.ch/student/stammt/doc/Inhalt.html` übernommen, teilweise überarbeitet.

2.1.1 Ethernet

Ein Ethernet-Paket ist wie folgt aufgebaut:



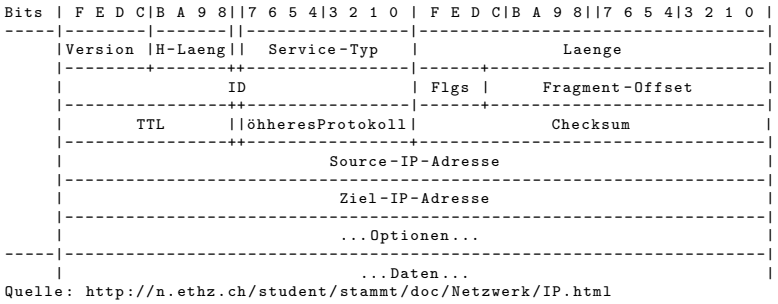
Quelle: http://n.ethz.ch/student/stammt/doc/Netzwerk/Ethernet.html

Zuerst wird die so genannte Preamble übertragen, welche aus 7 Bytes mit dem Bitmuster 10101010 (Hexadezimal 0xAA) und einem Byte mit Muster 10101011 (0xAB) besteht. Dieses Preamble wird benutzt, um den Empfänger zu synchronisieren. Wenn die Preamble defekt ist, kann davon ausgegangen werden, dass das restliche Paket ebenfalls unleserlich ist. Bei ordnungsgemäßer Übertragung folgen jetzt die beiden 6-Byte-(48-Bit)-MAC-Adressen für das Ziel und die Quelle. Daraufhin folgen 16 Bit mit dem Typ des nächst höheren Protokolls (z.B. IP). Nun werden die eigentlichen Nutzdaten übertragen. Zum Schluss wird noch eine 16-Bit-CRC-Prüfsumme angehängt.

2.1.2 IPv4

IP ist die Abkürzung für "Internet-Protokoll". Es ist zuständig für das korrekte versenden von Daten von einem Punkt zu einem anderen. Die Endpunkte werden dabei durch IP-Adressen angegeben. Diese Adressen bestehen aus 4 Byte-Zahlen, wie zum Beispiel 233.144.0.111.

Der Header des IP-Protokolls ist folgendermasse aufgebaut:



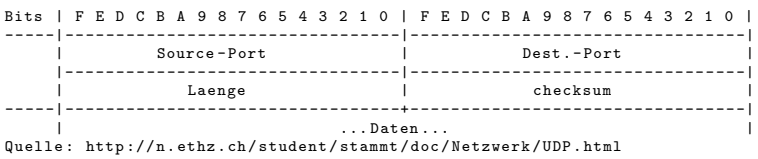
Quelle: http://n.ethz.ch/student/stammt/doc/Netzwerk/IP.html

- Version (4 Bit): Gibt an, um welche Version von IP es sich handelt (Momentan normalerweise v4).
- H-Laenge (4 Bit): Die Header-Länge gibt an, wieviele 32-Bit-Zeilen der gesamte Header besitzt (wird für zusätzliche Optionen benutzt).
- Service-Typ (8 Bit): Gibt an, um welche Art von Daten es sich handelt.
- Laenge (16 Bit): Gibt an, wieviele Bytes das gesamte Paket enthält.
- ID (16 Bit): Gibt an, um welches Paket es sich handelt.
- Flgs (3 Bit): Flags, werden für die Fragmentierung benutzt.
- Fragment-Offset (13 Bit): Gibt an, an welcher Stelle das Fragment eingefügt werden muss (in 8 Bytes).
- TTL (8 Bit): Die Time-To-Live gibt an, wieviele Hops das Paket noch weitergesendet werden darf.

- höheresProtokoll (8 Bit): Gibt an, welches Protokoll im nächst höheren Layer gebraucht wird.
- Checksum (16 Bit): Hier wird die Checksumme gespeichert (wird zu Korrektheitsüberprüfung gebraucht).
- Source-IP-Adresse (32 Bit): Gibt an, von welcher IP-Adresse das Paket kam.
- Ziel-IP-Adresse (32 Bit): Gibt an, zu welcher IP-Adresse das Paket gesendet werden soll.

2.1.3 UDP

Das "User Datagram Protocol", welches verbindungslos arbeitet, ist sehr einfach aufgebaut. Es erlaubt Daten in Form von einzelnen und unahbängigen Datergrammen über ein Netzwerk zu schicken. Die Anwendungsschicht muss sicherstellen, dass alle Pakete angekommen sind und in der richtigen Reihenfolge vorliegen. Das einzige was UDP prüft ist die Paketintegrität an Hand der Prüfsumme. Ein UDP-Paket ist folgendermassen aufgebaut:

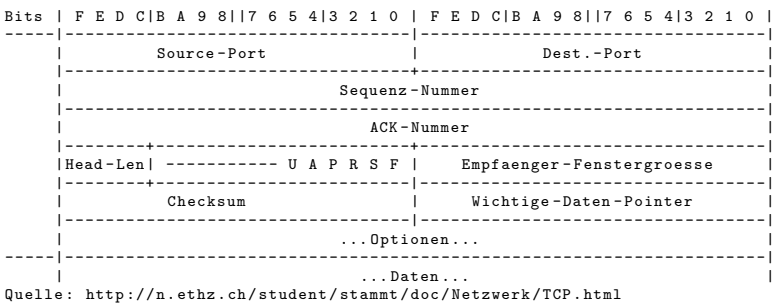


Quelle: http://n.ethz.ch/student/stammt/doc/Netzwerk/UDP.html

- Source- und Destination-Port (je 16 Bit): Geben an, von welchem Port sie gesendet wurden, und auf welchen Port sie gesendet werden.
- Laenge (16 Bit): Gibt die Anzahl Bytes (Maximal 65535) des Datenblockes an.
- Checksum (16 Bit): Ist eine Zahl, die durch die Addition der Einerkomplemente aller 16-Bit-Zahlen des Headers (ohne die Checksum) errechnet wird.

2.1.4 TCP

Das "Transmission Control Protocol" ist speziell entwickelt worden, um große Dateien in kleinere Pakete zu unterteilen und diese mittels Pipelining über ein Netzwerk zu senden. Standardisiert ist TCP durch RFC 793, 1122, 1323, 2018 und 2581. TCP-Pakete sind folgendermassen aufgebaut:



Quelle: http://n.ethz.ch/student/stammt/doc/Netzwerk/TCP.html

- Source- und Destination-Port (je 16 Bit): Geben an, von welchem Port sie gesendet wurden, und auf welchen Port sie gesendet werden.
- Sequenz-Nummer (32 Bit): Gibt an, welche Byteposition das Paket beschreibt.
- ACK-Nummer (32 Bit): Gibt an, welche Nummer das ACK-Signal besitzt.
- Head-Len (4 Bit): Gibt an, wieviele Zeilen (32 Bit) der Header enthält (wird benutzt für Optionen).
- Flags:
  - U: Urgent Data, ist gesetzt, falls in dem Datenblock wichtige Daten enthalten sind (wird aber grundsätzlich nicht benutzt).
  - A: ACK valid, ist gesetzt, falls die ACK-Nummer gültig ist.
  - P: Push data now, ist gesetzt, falls ??? (wird grundsätzlich nicht benutzt).
  - R, S, F: RST, SYN, FIN, wird benutzt, um die Verbindung auf- und abzubauen.
- Empfaenger-Fenstergroesse (16 Bit): Enthält die Anzahl Bytes, die der Empfänger akzeptieren wird.
- Checksum (16 Bit): Ist eine Zahl, die durch die Addition der Einerkomplemente aller 16-Bit-Zahlen des Headers (ohne die Checksum) errechnet wird.
- Wichtige-Daten-Pointer (16 Bit): Der Zeiger auf die so überaus wichtigen Daten (falls das U-Flag gesetzt ist).

Die Sequenz- und ACK-Nummer wird wie folgt verwendet: Wenn der Client ein Paket sendet, so enthält Sequenz-Nummer die Byte-Position innerhalb der gesamten Datei. ACK enthält einen Kontrollwert. Wenn nun der Server das Paket korrekt erhalten hat, so sendet er ein ACK-Paket zurück, welches als ACK-Wert das nächste zu sendende Byte (errechnet mittels Byte-Position plus Datenblock-Länge) enthält und als Sequenznummer den Kontrollwert von vorher. Dadurch kann der Client sehen, dass das Paket nun abgehakt werden kann, und er sieht zudem, ab welcher Byteposition er als nächstes senden muss.

3 Versuchsaufbau

Um die Performance des Netzwerkes möglichst direkt zu messen, wie gefordert, kam ein spezielles Testprogramm zum Einsatz: Netperf. Dieses Programm setzt direkt auf TCP bzw. UDP Sockets auf sendet bzw. empfängt Daten und misst die Zeit, die das Netzwerk braucht um diese Daten zu übertragen. Daraus wird dann die Datenübertragungsrate bzw. Geschwindigkeit berechnet.

3.1 Verwendete Computer

Es kamen zwei PCs zum Einsatz, welche folgende Charakteristiken aufwiesen:

- CPU Typ: AMD Athlon XP, 1544 MHz (5.75 x 269) 1800+
- Motherboard: Asus A7V266-EX (5 PCI, 1 AGP Pro, 1 ACR, 3 DIMM)
- Motherboard Chipsatz: VIA VT8366A Apollo KT266A
- Systembus: PCI-32 mit 33MHz
- Arbeitsspeicher: 128 MB (DDR SDRAM)
- BIOS Typ: Award Medallion (01/23/02)
- Grafikkarte: 3D Prophet 4000XT TV Out (32 MB)
- 3D-Beschleuniger: ST PowerVR Kyro
- Soundkarte: C-Media CMI8738 Audio Chip
- Diskettenlaufwerk
- Festplatte (40 GB, 7200 RPM, Ultra-ATA/100)
- ATAPI CD-R/RW 24X10 (24x/10x/40x CD-RW)
- PIONEER DVD-ROM DVD-116 (16x/40x DVD-ROM)

3.1.1 Netzwerkkarten

- Hersteller: 3Com
- Typ: 3Com Etherlink XL 3C905C-TX-M
- Geschwindigkeit: 10/100 MBps
- Bus: PCI

- Funktionallität: IEEE 802.1p (Prioritätsvergabe für Datenübertragung)
- Sonstiges: Passive (d.h. kein Prozessor auf der Netzwekkarte)
- IPs: 194.94.26.55 und 194.94.26.56
- MACs: 00:04:76:D6:83:A6 und 99:04:76:D6:83:A4

3.1.2 Betriebssysteme

GNU Linux

- Distribution: SuSE 9.0
- Kernel: `Linux 2.4.21-199-athlon #1 Fri Mar 12 08:24:04 UTC 2004 i686 athlon i386`
- Socks-Implementation: BSD Sockets

Windows

- Microsoft^{tm} Windows 2000 Professional
- Codename: Cairo
- Service Pack 4
- Sprache: Deutsch (Deutschland)
- Kerneltyp: Uniprocessor Free
- Version: 5.0.2195 (Win2000 Retail)
- Treiber für Netzwerkkarte: EL90BC
- Socks-Implementation: WinSocks (WINSOCK.DLL bzw. WSOCK32.DLL)

4 Versuchsdurchführung

Es wurden zwei Computer (PCs) über ein CAT-5 Cross-Over-Cable verbunden. Auf dem Server wurde jeweils netserver gestartet. Auf dem Client wurde jeweils netperf gestartet, welchem die Parameter für die aktuelle Messung übergeben wurden.

4.1 Netperf

Version unter Linux:

```
netperf.c (c) Copyright 1993-2003 Hewlett-Packard Company. Version
2.2pl3 netserver.c (c) Copyright 1993-2003 Hewlett-Packard Co. Ver-
sion 2.2pl3
```

Parameter von Server (netserver) Netserver wird ohne Parameter aufgerufen. Unter Linux geht das Programm selbstständig in den Hintergrund und läuft solange bis es (z.B. mit kill) abgeschossen wird. Unter Windows muss des Programm bei jeder Messung neu gestartet werden.

Parameter von Client (netperf) netperf wurde wie folgt aufgerufen:

```
netperf -H <ip> [-t UDP_STREAM] [-P0] - - -s <l.b.> -S <r.b.> -m <p.s.>
```

ip:	IP-Adresse des Zielgerätes auf dem netserver läuft
-t UDP_STREAM	UDP-Pakete senden. Default: TCP-Pakete.
-P0:	gibt nur die Messwerte aus und unterlasse die Ausgabe des Headers.
l.b.:	Local Send/Receive Buffer Size von 16, 20, 24, ..., 64kByte
r.b.:	Remote Send/Receive Buffer Size von 16, 20, 24, ..., 64kByte
p.s.:	Paket Größe: 64, 576, 1500, 65536 Byte

4.2 Messreihen

Es wurden vier Betriebssystem-Combinationen getestet:

Kennung	Server	Client
LL	GNU Linux	GNU Linux
WL	Windows	GNU Linux
LW	GNU Linux	Windows
WW	Windows	Windows

In jeder Kombination wurden jeweils acht Messreihen mit den Paketgrößen 64, 576, 1500, 65536 Byte und TCP oder UDP durchgeführt. Da UDP keine 65536 Byte großen Pakete verarbeiten kann, wurden hier 64512 Bytes verwendet.

In jeder Messreihe wurden die Sende- und Empfangepuffer jeweils gleich auf folgende Werte eingestellt:

1. 16384 Byte = 16 kByte
2. 20480 Byte = 20 kByte
3. 24576 Byte = 24 kByte
4. 28672 Byte = 28 kByte
5. 32768 Byte = 32 kByte
6. 36864 Byte = 36 kByte
7. 40960 Byte = 40 kByte
8. 45056 Byte = 44 kByte
9. 49152 Byte = 48 kByte
10. 53248 Byte = 52 kByte
11. 57344 Byte = 56 kByte
12. 61440 Byte = 60 kByte
13. 65536 Byte = 64 kByte

4.3 Bemerkungen

Die Cross-Over-Verbindung über das Patchfeld funktionierte nicht. Es wurde daher ein "normales" 2m Cross-Over-Kabel verwendet.

Alle Messungen liefen 10 Sekunden. Dies ist die default-Einstellung bei Netperf. Die unter GNU Linux durchgeführten Messungen hielten sich mit 10 bis 10.01 Sekunden ziemlich genau an die Vorgaben. Unter Windows variierte die Messdauer zwischen 11 und 19 Sekunden.

Die Messungen zwischen den Windows-Computern (WW) gestaltete sich leider sehr schwierig. Wir hatten mit verdächtig langsam laufenden Computern sowie mit massieven Systemausfällen zu kämpfen, so dass wir die Messung, welche bereits auf einen dritten Labortermin gelegt werden mußte, zum Schluß aufgeben mußten. Die Messungen an denen Windows nur zum Teil beteiligt war liefen bis auf einige Ausfälle vergleichsweise gut.

5 Versuchsergebnisse

LL – Server: GNU Linux; Client: GNU Linux									Kombination
Buffer	TCP				UDP				Protokoll
größe	64	576	1500	65536	64	576	1500	65536	Paketgröße
16	88.35	93.80	93.93	94.06	48.51	89.59	92.27	95.90	Messungen
20	89.93	93.67	93.88	93.94	48.45	89.64	92.03	95.79	
24	90.79	93.85	94.03	94.05	48.49	89.68	92.24	96.05	
28	91.12	93.91	94.07	94.05	48.58	89.68	92.25	96.05	
32	93.89	93.93	94.09	94.06	46.86	89.69	92.27	96.05	
36	93.93	93.96	94.12	94.07	48.51	89.70	92.28	96.05	
40	93.97	93.97	93.97	94.06	48.46	89.71	92.29	96.10	
44	93.99	93.99	94.00	94.03	48.38	89.72	92.31	96.10	
48	93.99	93.99	94.01	94.10	45.40	89.67	92.28	96.15	
52	94.01	94.04	94.04	94.09	45.58	89.67	92.29	96.15	
56	94.02	94.03	94.06	94.09	44.29	89.67	92.30	96.10	
60	94.02	94.04	94.07	94.10	43.10	89.66	87.14	96.15	
64	94.04	94.04	94.09	94.10	41.58	89.67	87.14	96.15	
ϕ	92.77	93.94	94.03	94.06	46.63	89.67	91.47	96.06	Durchschnitt

WL – Server: Windows; Client: GNU Linux									Kombination
Buffer	TCP				UDP				Protokoll
größe	64	576	1500	65536	64	576	1500	65536	Paketgröße
16	93.45	93.83	93.95	94.02	02.57	89.40	92.35		Messungen
20	93.78	93.96	93.96	94.11	02.53	89.71	92.36		
24	93.94	93.93	94.02	94.12	02.04	89.58	92.37	56.31	
28	93.91	93.91	94.01	94.12	02.71	89.72	92.38	96.15	
32	94.08	94.13	94.13	94.01	02.48	89.68	92.37	95.94	
36	93.94	93.83	94.05	94.10	02.79	89.72	92.38	96.20	
40	93.97	93.97	93.97	94.12	02.63	89.73	92.38	96.20	
44	93.99	93.98	94.01	94.12	02.48	89.73	92.40	96.20	
48	94.13	94.12	94.12	94.13	02.52	89.75	92.38	96.20	
52	94.12	94.13	94.13	94.13	02.58	89.75	92.38	96.20	
56	94.02	94.05	94.06	94.12	02.62	89.75	92.21	96.20	
60	94.11	94.12	94.13	94.13	02.61	89.75	01.89	96.20	
64	94.13	94.13	94.13	94.13	02.60	89.75		96.20	
ϕ	93.97	94.01	94.05	94.10	2.55	89.69	84.82	92.55	Durchschnitt

LW – Server: GNU Linux; Client: Windows									Kombination
Buffer	TCP				UDP				Protokoll
größe	64	576	1500	65536	64	576	1500	65536	Paketgröße
16	90.44	91.79	90.54	95.00	38.29	76.92	65.38		Messungen
20	88.61	91.75	90.61	94.95	38.65	77.59	65.11		
24	84.19	91.74	90.63	95.00	38.74	77.83	65.15		
28	91.36	91.86	90.55	95.00	38.77	77.88	65.40		
32	89.83	91.81	90.55	95.00	38.75	78.19	65.10		
36	89.41	91.98	90.63	94.95	38.86	78.56	65.17		
40	82.98	91.89	90.64	95.00	39.00	78.37	65.16	92.54	
44	87.51	91.95	90.64	95.00	39.43	79.90	65.21	92.54	
48	87.13	91.98	90.64	94.95	39.15	80.36	65.14	92.48	
52	87.19	91.94	90.65	95.00	39.30	80.55	65.17	92.48	
56	90.98	91.93	90.62	95.00	39.12	80.29	65.07	92.54	
60	78.66	91.91	90.63	94.95	39.39	81.14	65.04	92.54	
64	89.80	94.43	93.97	95.00	39.59	81.11	65.02	92.48	
ϕ	87.55	92.07	90.87	94.98	39.00	79.13	65.16	92.51	Durchschnitt

WW – Server: Windows; Client: Windows									Kombination
Buffer	TCP				UDP				Protokoll
größe	64	576	1500	65536	64	576	1500	65536	Paketgröße
16	0.20	0.21	0.21	0.19					Messungen
20	0.21	0.21	0.21	0.24					
24	0.21	0.21	0.20	0.14					
28	0.20	0.22	0.20	0.19					
32	0.20	0.22	0.20						
36	0.20	0.22	0.21						
40	0.21	0.19	0.21						
44	0.19	0.21	0.21						
48	0.21	0.20	0.21						
52	0.20	0.22	0.20						
56	0.20	0.20	0.20						
60	0.20	0.20	0.19						
64	0.21	0.21	0.21						
ϕ	0.20	0.21	0.20	0.19	0.00	0.00	0.00	0.00	Durchschnitt

Übersich über die Durchschnittswerte der Messungen:

Protokoll:	TCP				UDP			
Paketgröße:	64	576	1500	65536	64	576	1500	65536
LL	92.77	93.94	94.03	94.06	46.63	89.67	91.47	96.06
WL	93.97	94.01	94.05	94.10	02.55	89.69	84.82	92.55
LW	87.55	92.07	90.87	94.98	39.00	79.13	65.16	92.51
WW	00.20	00.21	00.20	00.19				

Erstaunlich ist, dass die Kombination Server: Windows und Client: Linux im Schnitt etwas schneller bei der TCP-Übertragung war, als Linux mit Linux. In allen anderen Kombinaionen hatte Linux mit Linux die Nase vorn. Bei der UDP-Übertragung und bei kleinen Paketgrößen deutlicher als bei TCP.

Bei der Datenübertragung der Windows-zu-Windows-Messung ging garnichts mehr,

und dass an beiden Laborveranstaltungen, an denen die Messung versucht wurde. Die Messergebnisse aus dieser Messung sind unrealistisch klein. Im Schnitt 0.2MBps auf einer 100MBps-Strecke sind etwas wenig.

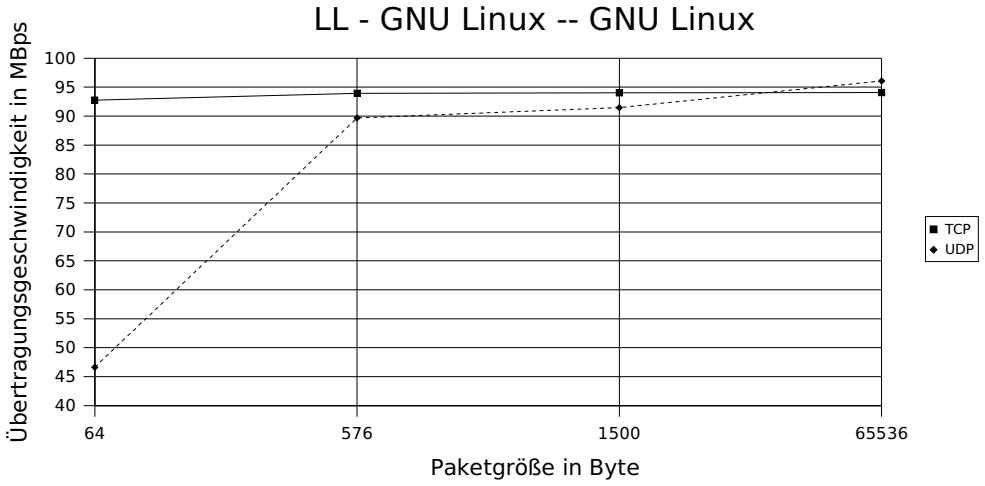
6 Versuchsauswertung

Da die Messungen an denen Microsoft^{tm}-Produkte beteiligt waren Probleme bereiteten und unvollständig sind, verwende ich hier die Messergebnisse, welche mit Hilfe von Linux gewonnen wurden.

6.1 Auswirkung der Paketgröße

Wie erwartet stieg die Übertragungsgeschwindigkeit mit steigender Paketgröße. Dies ist darauf zurückzuführen, dass kleine Paketen viel Overhead verursachen.

Wenn also 100 Einheiten Daten übertragen werden sollen, können diese in 100 Paketen zu je einer Dateneinheit, in 10 Paketen zu je 10 Dateneinheiten oder in 1 Paket mit 100 Dateneinheiten verschickt werden. Wenn man nun annnimmt, dass je Paket eine Dateneinheit zur Verwaltung hinzu kommt, ist schnell zu sehen, dass der Overhead bei den kleinen Paketen 100 Dateneinheiten beträgt, also 50%. Im besten Fall, mit 100 Dateneinheiten in einem Paket, liegt er bei einer Dateneinheit, was etwas weninger als einem Prozent entspricht.



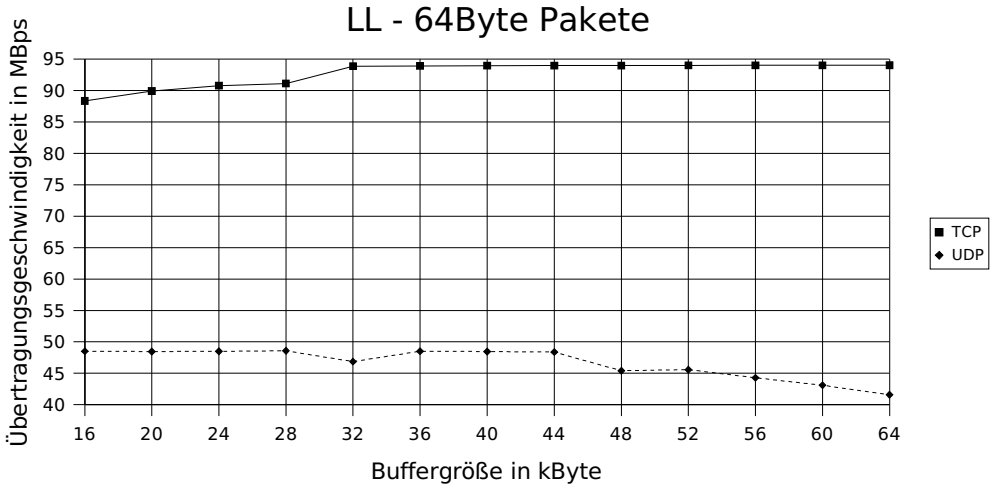
In diesem Diagramm wurden die Mittelwerte aus der Linux zu Linux - Messung dargestellt. Es ist zu erkennen, dass bei steigenden Paketgrößen mehr Daten in weniger Zeit übertragen wurden – die Datenübertragungsrate also mit steigender Paketgröße steigt. Bei der TCP-Messung ist diese Entwicklung nur schwach zu erkennen. Die Übertragungsrate steigt dort von 92.77 MBps bei 64Byte Paketgröße auf 94.06MBps bei 65kByte Paketen. Bei UDP-Paketen ist diese Entwicklung wesentlich ausgeprägter.

Das ist, von der Protokollseite her betrachtet, eher verwunderlich, da TCP-Pakete

einen wesentlich größeren Header haben. Anscheinent ist die TCP-Verarbeitung bei Linux leistungsstärker bei kleinen Paketen als die UDP-Verarbeitung, was diesen Effekt erklären könnte.

6.2 Auswirkung der Puffergrößen

Die Auswirkung der eingestellten Sende- und Empfangspuffer ist lediglich bei den 64-Byte-Paketen zu erkennen. Bei den anderen Paketgrößen sind keine nennenswerten Auswirkungen zu erkennen.



Wie in dem Diagramm zu erkennen ist, steigt die Leistung bei TCP mit größerwerdenen Puffern. Bei UDP verhält sich dies genau anders herum. Eine mögliche Erklärung ist, dass die TCP-Pakete in einem Stream gesendet wurden. Also Netperf einen TCP-Stream öffnete und Pakete in diesen sendete. So konnte der Puffer gut ausgenutzt werden, was zu weniger Zugriffen von Netperf zum Betriebssystem führte. Die Sättigungsgrenze war hier allerdings schnell erreicht.