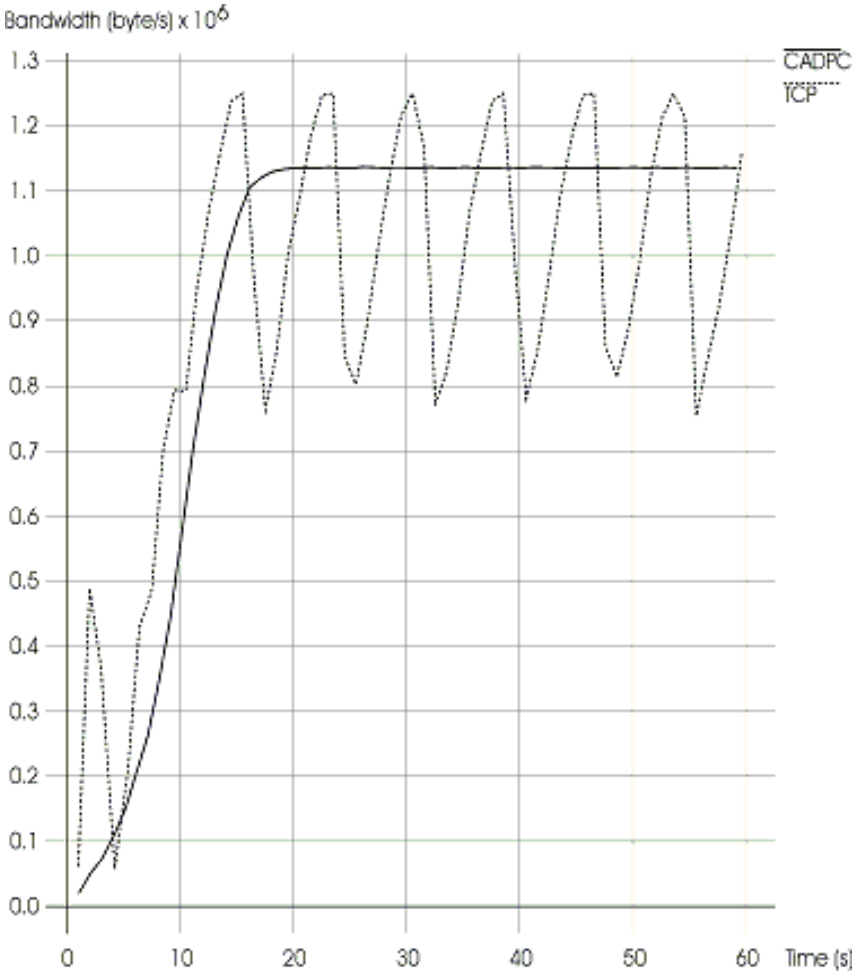
Aktualisierung des PTP-Protokolls



Name: Martin Rabanser

Matrikelnr.: 0217224

Datum: 06.12.2005

Tutor: Michael Welzl

Inhaltsverzeichnis

1. Einführung

1.1 Problemstellung

Aufgabe diese Bakkalaureatsarbeit war es, die bestehende Implementierung des PTP-Protokolls auf einen aktuellen Stand zu portieren, sodass an dessen Entwicklung weitergearbeitet werden kann. Hierzu war es nötig den bestehenden Kernelpatch für einen aktuellen Kernel um zuschreiben und anschließend die neue Spezifikation des Protokolls zu implementieren. Gleichzeitig sollte auch der bestehende Code etwas gesäubert werden, damit nicht mehr benötigte Zeilen den Code nicht unnötig unübersichtlich machen.

1.2 Das Performance Transparency Protocol (PTP)

Bei Netzwerkverbindungen bei denen ständig viele Daten übertragen werden, beispielsweise Streaming, Videotelefonie etc. kommt es vor, dass die Übertragungsrate entweder grösser oder kleiner als die Kapazität eines Routers sind. Im ersten Fall hat die Verbindung einen Flaschenhals und Pakete werden verworfen. Im zweiten Fall wird die Verbindung nicht optimal ausgenutzt da mehr Pakete als nötig gesendet werden und auf der Leitung fließt automatisch mehr Overhead als nötig. Genau an diesem Punkt soll das PTP-Protokoll eingreifen und die Übertragungsrate an die größtmögliche Bandbreite auf einem Netzwerkpfad anpassen.

Das PTP-Protokoll wurde von Michael Welzl spezifiziert und soll dazu beitragen den Netzwerkverkehr zu optimieren und Router optimal auszulasten indem Daten über deren Geschwindigkeit gesammelt und ausgewertet werden. Dabei werden Bandbreiteninformationen von jedem Router auf einem Netzwerkpfad an Pakete gehängt, welches am Endknoten die Grundlage für die Performanceberechnungen darstellen. Anschließend werden die errechneten Daten dem Sender zurückgeschickt, welcher angemessen auf die Informationen reagieren kann. Ziel

ist es, dieses Teilprotokoll mit einem bestehenden Protokoll z.B. UDP zu kombinieren, sodass der Anwender (Anwendung) auf höherer Schicht von der Optimierung nichts bemerkt und das Protokoll eigenständig und automatisch ein gesamtes Teilnetz optimiert.

1.3 Einsatzgebiete

Das PTP-Protokoll wurde dazu entwickelt, die Übertragungsraten für Applikationen an die Gegebenheiten des Pfades anzupassen. Ein aktuelles Beispiel dafür wäre die Videotelefonie.

Ein weiteres Einsatzgebiet wäre das PTP-Protokoll als Administrationstool für eine Domäne, in dem ein Administrator die Kontrolle über all seine Rechner hat. So könnte das Protokoll dazu eingesetzt werden, um dem Administrator zu signalisieren an welchen Stellen sein Netzwerk besonders ausgelastet ist bzw. wo es Flaschenhälse gibt. Da dies aber nicht das verfolgte Ziel bei der Spezifikation des Protokolls war, kann es für dieses Szenario durchaus notwendig sein, einige Anpassungen an der Implementierung vorzunehmen.

Grundsätzlich kann dieses Protokoll als Grundlage für Staukontrolleverfahren wie z.B. CADPC eingesetzt werden, welches weniger Paketverluste, teilweise höhere und glattere Übertragungsraten als TCP erreicht.

1.4 Die Teststrecke

Das Testen des PTP-Protokolls ist mit einigem Aufwand verbunden da 2 Endknoten nicht genügen, sondern ein Netzwerk mit mindestens 3 Rechner zum Einsatz kommen, wobei jene in der Mitte als Router fungiert.

Der Aufwand für den Betrieb der beiden Endknoten ist relativ gering. Es genügt die Installation eines beliebigen Linuxsystems, Voraussetzungen sind lediglich ein funktionierendes Netzwerk in der Version IPv4 und ein “gcc-Kompiler”. Dieser ist zwar nicht zwingend notwendig, aber die Kompilierung der Testprogramme auf

dem Zielrechner garantiert deren korrekte Funktion. Eventuell kann auch eine Knoppixdistribution zum Einsatz herangezogen werden, sollten keine Linuxrechner zur Verfügung stehen und/oder eine Neuinstallation zu aufwendig sein. Andere Betriebssysteme kommen derzeit nicht in Frage, da die Sockets mit dem Protokoll Nr. 123 initiiert werden, welches zwar offiziell bei der Authorisierungsstelle angemeldet wurde, aber sicher noch nicht in jedem Betriebssystem verwendet werden kann. Weiters greifen sowohl Client als auch Server auf einen Handler zurück der auf die posixkonforme IPC von Unix aufsetzt, welche bei weitem nicht von allen Systemen eingehalten wird.

Das Aufsetzen des PTP-Routers ist im Vergleich zu den Endknoten wesentlich aufwendiger. Voraussetzung hierfür sind ein beliebiges Linuxsystem sowie der Einsatz des richtigen Kernels und 2 Netzwerkinterfaces. Verwendet wird hier der Standard Vanillakernel dessen Versionsnummer mit der des PTP-Patches übereinstimmt, sodass er vor seinem Einsatz erfolgreich gepatcht werden kann. Bei Verwendung anderer Konfigurationen ist keinerlei Funktion gewährleistet. Damit die Pakete richtig geroutet werden muss der Kernel gepatcht und mit allen wichtigen Routerfunktionen übersetzt werden (IP-Forward). Anschließend wird dieser Kernel gestartet, die Netzwerkinterfaces richtig initialisiert und manuell die Bandbreite des aktuellen Router eingestellt, damit das PTP-Protokoll Daten sammeln und auswerten kann.

2. Patchen des Kernels

2.1 Anpassung des bestehenden Patches

Für die Anpassung des Patches an einen aktuellen Kernel, musste vorerst der bestehende Patch eines SuSE-2.2.10-Kernel für den Standardkernel 2.2.10 angepasst werden um eine standardisierte Ausgangsposition zu schaffen. Distributionen pflegen das Einfügen von Zusatzpatches in deren Kernel, welche

Probleme hervorrufen können. Weiters ist es für den Anwender von PTP, zu einem späteren Zeitpunkt, schwieriger einen solchen proprietären Kernel zu finden, als einen Standard-Vanillakernel der jederzeit in den Archiven von www.kernel.org zu vorzufinden ist.

Die für den Patch relevanten Dateien differenzierten sich in den beiden Kernelversionen nur an einer Stelle. Die Zeile 1142 in der Datei `linuxkernel/net/ipv4/af_inet.c`, in die der Patch folgenden Code eingefügt, unterscheidet sich um ein Offset von -11 auf 1131[1]:

```
diff -Naur linux-2.2.10.SuSE.orig/net/ipv4/af_inet.c linux-
2.2.10.SuSE/net/ipv4/af_inet.c
--- linux-2.2.10.SuSE.orig/net/ipv4/af_inet.c Thu Jul 22
16:05:48 1999
+++ linux-2.2.10.SuSE/net/ipv4/af_inet.c  Tue Jun  6
11:37:34 2000
@@ -1142,6 +1142,14 @@
@@ -1131,6 +1131,14 @@
 #ifdef CONFIG_INET_RARP
  rarp_ioctl_hook = rarp_ioctl;
 #endif
+
+ /*
+  *  set the ptp layer up
+  */
+#ifdef CONFIG_IP_PTP
+ ptp_init(&inet_family_ops);
+#endif
+
  /*
   *  Create all the /proc entries.
   */
```

Der Einsatz des gepatchten Kernels 2.2.10 erfordert eine geeignete Linuxumgebung, mit versionsgemäßen Compilern. Auf weitere Entwicklung und Testen mit dem alten Kernel wurde verzichtet, da das Ziel des Projektes die Anpassung des Protokolls auf einen neuen Kernel ist. Deshalb wurde unmittelbar

1 Rot steht für SuSE-Kernelpatch linux-2.2.10-SuSE.ptp.patch, Blau für Vanillakernelpatch linux-2.2.10.ptp.patch

zur Portierung des Patches auf einen aktuelleren Kernel übergegangen.

2.2 Wahl einer geeigneten aktuellen Kernelversion

Zur Zeit dieser Entscheidung befand sich der Kernel 2.6 gerade in der letzten Testphase bzw. wurde schon mit einigen Distributionen ausgeliefert. So lag es nahe sich für den letzten stabilen Kernel der Version 2.4 mit Versionsnummer 2.4.26 zu entscheiden, der eine weite Verbreitung genoss. Begründet wird diese Entscheidung vor allem dadurch, dass dieses Protokoll auf Routern eingesetzt werden soll, bei denen mehr Wert auf Sicherheit als auf Hardwareunterstützung und somit Aktualität gelegt wird.

Zur Zeit der Entstehung dieses Dokumentes existieren auch für Kernel 2.6 stabile Versionen, sodass eine Portierung auf diesen in Anbetracht gezogen werden kann.

2.3 Manuelles Einfügen des PTP-Codes

Da testweises Patchen des Kernel 2.4.26 mit dem Patch linux-2.2.10-ptp.patch mit unzähligen Fehlern abbrach, musste jede Zeile des Patches für den Kernel 2.2.10 manuell in den Quellcode des Kernels 2.4.26 eingefügt werden. Die Schwierigkeit bestand zum Einen darin die richtigen Stellen zu finden, da sich die Netzwerkimplementierung des IP-Protokolls v4 von Kernel 2.2 auf 2.4 sehr stark geändert hatte. Weit aufwendiger war das Verstehen des Codes und die Suche von Funktionen die es von Version 2.2 auf Version 2.4 entweder nicht mehr gab oder aber deren Datentypen oder Aufrufe stark verändert aufwiesen. Im Folgenden werden diese kritischen Stellen im Einzelnen genau beschrieben. Auf die Angabe der Zeilenänderungen wird vorerst verzichtet, da der Kernel manuell gepatcht wurde und die richtigen Codestellen ohnehin gesucht werden mussten. Die Angaben der Zeilennummern werden später im Kernelpatch automatisch mitprotokolliert.

Die Struktur `struct device` in Datei

`linuxkernel/include/linux/netdevice.h` heisst in Version 2.4 `struct net_device` in welcher der Zeiger `void *ip_ptr` relevant ist. Nach Überprüfung auf die Gleichheit der beiden Strukturen, wurde der Name der Struktur in der Datei `linuxkernel/include/net/ptp.h` einmal und viermal in Datei `linuxkernel/net/ipv4/ptp.c` angepasst.

In der alten Version des PTP-Codes wurde eine Funktion Namens `skb_datarefp()` verwendet, welche in Datei `linuxkernel/include/linux/skbuff.h` an Zeile 165 definiert war:

```
extern_inline_atomic_t * skb_datarefp (struct sk_buff *skb)

{

     return (atomic_t *) (skb_end);

}
```

Diese Funktion beinhaltet keine aufwendige Implementierung sondern liefert lediglich die Position des Letzten Elementes einer Liste zurück. Weil an der Struktur `struct skb` jedoch nichts verändert wurde, lag es nahe diesen Aufruf einfach direkt in den PTP-Code einzubauen, sodass aus Zeile 109 in `linuxkernel/net/ipv4/ptp.c` von:

```
atomic_set(skb_datarefp(skb), 1);

atomic_set(((atomic_t *)(skb->end)),1);
```

wurde.

Die Funktion `get_fast_time ()` existiert in der neuen Version auch nicht mehr, jedenfalls nicht für die i386 Architektur. So wurde aus dem Funktionsaufruf `get_fast_time (&tv);` an Zeile 140 `do_gettimeofday (&tv);` welcher schon im alten Kernel existiert und genau dasselbe zurückliefert. Die Bezeichnung der Funktion passt besser zu ihrer Aufgabe, deshalb wurde sie wahrscheinlich im

alten Code verwendet. Diese neue Funktion ist zweimal implementiert und zwar in `linuxkernel/kernel/time.c` und in `linuxkernel/arch/i386/kernel/time.c`. Beidemal wird die Zeit `xtime` zugewiesen, es ist daher unkritisch diese Funktion zu verwenden.

Weiters hatte die Funktion __initfunc () an Zeile 397 in `linuxkernel/net/ipv4/ptp.c` keinen Rückgabewert. Dies korrigiert das Anfügen von `void` vor der Funktion. Problematisch bei dieser Funktion ist der fehlerhafte Code, weswegen sie nicht verwendet wird. Sie wurde vollständig entfernt.

Das Element `ip_ttl` in der Struktur `struct sock` in Datei `linuxkernel/include/net/sock.h` wurde in die Struktur `struct inet_opt` ausgelagert und diese letztere in die Struktur `struct sock` eingefügt. Deshalb musste Zeile 413 von `linuxkernel/net/ipv4/ptp.c` von `ptp_socket->sk->ttl` nach `ptp_socket->sk->af_inet.ttl` angepasst werden.

Die letzte relevante Veränderung im Kernel 2.4 sind die Makefiles, bei denen nun die Syntax aufgrund der höheren „automake“-Version etwas anders ist. In Datei `linuxkernel/net/ipv4/Makefile` werden nun nicht mehr die Zeilen:

```
ifeq ($(CONFIG_IP_PTP),y)

IPV4_OBJS += ptp.o

endif
```

eingefügt, sondern einfach an Zeile 28 folgende Codezeile hinzugefügt:

```
obj-$(CONFIG_IP_PTP) += ptp.o
```

Alle anderen Codezeilen des PTP-Protokolls wurden nur an den richtigen, zum Teil verschoben, Zeilen eingefügt. Die Zeilennummern sind allein im Patch ersichtlich.

2.4 Kritische Stellen

Bei der Anpassung des Protokolls gab es einige Stellen die bis nach der Testphase als kritisch galten. Diese seien hier aufgeführt, auch weil diese bei einer späteren Portierung auf einen neueren Kernel Schwierigkeiten bereiten können.

In Datei `linuxkernel/include/linux/sysctl.h wurden` zwischen Zeile 348 und 368 bei folgendem Enum weitere Einträge angefügt, sodass die Konstante `NET_IPV4_CONF_IF_SPEED` nun den Wert 20 erhält:

```
enum

{

  NET_IPV4_CONF_FORWARDING=1,

  NET_IPV4_CONF_MC_FORWARDING=2,

  NET_IPV4_CONF_PROXY_ARP=3,

  NET_IPV4_CONF_ACCEPT_REDIRECTS=4,

  NET_IPV4_CONF_SECURE_REDIRECTS=5,

  NET_IPV4_CONF_SEND_REDIRECTS=6,

  NET_IPV4_CONF_SHARED_MEDIA=7,

  NET_IPV4_CONF_RP_FILTER=8,

  NET_IPV4_CONF_ACCEPT_SOURCE_ROUTE=9,

  NET_IPV4_CONF_BOOTP_RELAY=10,

  NET_IPV4_CONF_LOG_MARTIANS=11,

  NET_IPV4_CONF_TAG=12,

  NET_IPV4_CONF_ARPFILTER=13,
```

```
  NET_IPV4_CONF_MEDIUM_ID=14,
  NET_IPV4_CONF_FORCE_IGMP_VERSION=17,
  NET_IPV4_CONF_ARP_ANNOUNCE=18,
  NET_IPV4_CONF_ARP_IGNORE=19,
  NET_IPV4_CONF_IF_SPEED=20
};
```

Diese Zeile ist kritisch, da nicht sicher ist ob der Wert 20 in Zukunft von einer anderen Konstanten im Enum genutzt wird. Dies würde natürlich zu Inkonsistenzen führen, darum soll der Programmierer des PTP-Protokolls diese Stelle unbedingt im Hinterkopf behalten.

Weiters befindet sich in der Datei `linuxkernel/net/ipv4/devinet.c` auf Zeile 1138 die Deklaration des Feldes `ctl_table devinet_vars [20];` welches auf `ctl_table deinet_vars [21];` erweitert wird. Begründet dadurch, dass im Kernel 2.2.10 das Feld die Größe 12 hat und durch den Patch auf 13 vergrößert wird. Auch hier ist nicht sichergestellt, dass die Feldgröße nicht durch Kernelprogrammierer oder weitere Patches verändert wird.

Bemerkenswert ist, dass in derselben Datei wie oben, zwischen Zeile 1143 und 1199, nur 18 Einträge des Feldes initialisiert werden, obwohl das Feld eine Größe 21 aufweist. Dies sollte aber egal sein.

Weiters wurden die Zeilen 514-516 im Kernelpatch entfernt. Diese Überreste aus einer nicht mehr aktuellen Implementierung waren vorerst schuld daran, dass am Router die falsche Checksumme berechnet und infolgedessen die Pakete verschmissen wurden. Auch die Abfragen in den Zeilen 508-510 sind, genau genommen, überflüssig, vorausgesetzt das Feld „count“ im Header wird stets richtig gesetzt. Um die Stabilität des Codes etwas zu erhöhen bleiben diese jedoch erhalten.

Ein weiterer kritischer Punkt waren die Zeilen 261-275 in „handler.cc“. Bei einem Antwortpaket wurde davon ausgegangen, dass im Anhang die originale Zieladresse stünde. Da sich die Spezifikation davon jedoch weit entfernt hat und der Handler somit die empfangenen Daten an keine gültige Adresse zustellen konnte, wurden die Pakete verschmissen. Deshalb wurde dieser Codeteil vollständig entfernt.

2.5 Generieren des PTP-Patches

Nachdem alle Einträge des „alten“ Patches im neuen Kernel an deren richtigen Stellen eingefügt sind und die Kompilierung des manuell gepatchten Kernel keinen Fehler mehr wirft, kann die erste Version des neuen Kernelpatches generiert werden. Wie dieser erstellt wird, kann jederzeit im Patch selbst überprüft bzw. nachgeschlagen werden. Verwendet wird hierbei der Befehl `diff`, welcher Differenzen zwischen 2 Dateien findet. Voraussetzung sind 2 Verzeichnisse namens `linux-2.4.26`, welches den originalen entpackten Vanillakernel 2.4.26 enthält, und `linux-2.4.26-ptp`, welches einen entpackten und manuell gepatchten Kernel enthält. Vor dem Generieren des Patches muss sichergestellt sein, dass sich keine Objektdateien oder dergleichen in den Verzeichnissen aufhalten. Dies stellt der Befehl `make mrproper` sicher, welcher die Verzeichnisse „säubert“. Anschließend vergleicht der Befehl `diff -Naur linux-2.4.26/ linux-2.4.26.ptp/ > linux-2.4.26-ptp.patch` den Inhalt aller Unterverzeichnisse und Dateien des originalen Kernels mit jenen des manuell Gepatchten und schreibt die Differenzen in die Datei `linux-2.4.26-ptp.patch` mit.

Die vergleichsweise unkomplizierte Generierung des Patches ist somit abgeschlossen und erste Tests können gestartet werden. Bis zum endgültigen Patch sollten jedoch noch einige Veränderungen hinzukommen, sodass dieser bis zum Ende des Projektes noch einige Male neu erstellt werden sollte.

2.6 Installation & Test des PTP-Patches

Damit das PTP-Protokoll nun auf dem neuen Kernel getestet werden kann, muss der Kernel gepatcht werden. Zwar existiert schon ein manuell gepatchter Kernel, aber schließlich soll der Patch selbst auf dessen korrekte Funktion getestet werden. Der Befehl `patch -p0 < linux-2.4.26-ptp.patch`, ausgeführt im selben Verzeichnis wo er erstellt wurde (impliziert die Option -p0) auf einen frisch entpackten Vanillakernel 2.4.26, modifiziert die Quelldateien wie gewünscht.

Im nächsten Schritt wird der Kernel kompiliert, wobei darauf geachtet werden muss, dass die Option den PTP-Code mit zu übersetzen auch aktiviert ist. Damit diese Option anwählbar ist, bedarf es der Aktivierung des Schalters `Prompt for development and/or incomplete code/drivers` im Menüpunkt `Code maturity level options`. Anschließend kann unter `Networking Options` der Punkt `IP: PTP support (EXPERIMENTAL) (NEW)` angewählt werden. Zu Beachten ist weiters die grundsätzliche Aktivierung der Netzwerkunterstützung inklusive der Auswahl der richtigen Treiber für die Netzwerkhardware, sowie für IP-Forwarding.

Sobald der Kernel erfolgreich übersetzt und der Router mit diesem gestartet wurde, benötigt das Protokoll für einen funktionierenden Einsatz noch die Information über die Geschwindigkeit des Routers. Dies muss manuell im `/proc` -Verzeichnis gesetzt werden und wird von folgenden Codezeilen automatisiert, vorausgesetzt die Netzwerkschnittstellen zwischen in Verbindung mit den beiden Endknoten sind jeweils eth0 und eth1:

```
echo 10000 > /proc/sys/net/ipv4/conf/all/mib2_if_speed

echo 10000 > /proc/sys/net/ipv4/conf/default/mib2_if_speed

echo 10000 > /proc/sys/net/ipv4/conf/eth0/mib2_if_speed

echo 10000 > /proc/sys/net/ipv4/conf/eth1/mib2_if_speed
```

Hierbei wird der MIB2_IF_SPEED auf den Wert 10000 gesetzt. Dies ist kein reeller Wert und dient lediglich zum Testen.

Zur Überprüfung der korrekten Funktion des neuen PTP-Kernels werden die bereits bestehenden Client/Server-Programme herangezogen, die sich nach minimalen Anpassungen der einzubindenden Bibliotheken problemlos übersetzen und starten ließen.

3. Aktualisierung des PTP-Protokolls von v3.0 auf v5.0

3.1 Grundsätzliche Funktionsweise

a) Server

Der Server des Testprogrammes ist hier verwirrenderweise nicht jener der die Daten liefert sondern sie vom Client abfragen wird. Grundsätzlich ist die Aufgabe des Servers jene, die gewünschten Anfragen von der Benutzerschnittstelle entgegenzunehmen, in eine korrekte PTP-Anfrage zu verfassen und darauf zu warten, dass der Client die ausgewerteten Daten in einer Feedbackmessage zurückschickt, welches interpretiert und gespeichert wird. Diese Daten können dann über definierte Schnittstellen abgefragt werden, beispielsweise von einem UDP-Protokoll.

Die öffentlichen Funktionen sind in der Datei `ptpserver.h` definiert, wobei die interessanteste wahrscheinlich `get_bandwidth(...)` (mit entsprechenden Argumenten) ist.

b) Client

Der Client liefert als Gegenstück zum Server wie erwähnt die angeforderten Informationen über den Pfad vom Server zum Client. Seine Aufgabe ist es, aus der Liste der Rohdaten, die an den Routern gesammelt werden, alle Werte auszulesen und auszuwerten. Hier werden sowohl die Informationen über einen möglichen

Flaschenhals als auch Aussagen über den Traffic auf einer Route errechnet. In der alten Implementierung wurde hier der Trend berechnet.

c) Handler

Der Handler ist das Verbindungsstück zwischen Netzwerk und Benutzerschnittstelle und wird sowohl auf dem Server sowie Client verwendet. Er läuft im Hintergrund und wartet auf Daten an der angegebenen Netzwerkschnittstelle. Nach Überprüfung der Daten sowie der Gegenstelle puffert er das Paket und lässt es per IPC (Inter Process Communication) vom Server bzw. Client auslesen.

Über Sinn oder Unsinn dieser relativ aufwendigen Implementierung lässt sich diskutieren, gehört aber zum bereits bestehenden Code dazu und soll nicht weiter Thema dieses Projektes sein. Begründet dadurch, dass der Code fast vollständig kompatibel mit der neuen Umgebung ist und daher nur gering verändert werden muss.

d) Router

Bisher wurde nur über das Patchen des Kernels gesprochen nicht aber über die eigentliche Aufgabe des Routers. Unabhängig davon wieviele Router zum Einsatz kommen, automatisiert das Protokoll das Sammeln der Daten vollständig und jeder Router ist ,eng betrachtet, identisch. Nach dem Starten von Server und Client auf den Endknoten und Pakete vom Server zu seinem Client weitergeroutet werden fügen die Router deren Informationen in den Anhang der Pakete. Hierbei werden die eingestellten Optionenam Server berücksichtigt, sodass die richtigen Daten im Anhang Platz finden. Auf der Rückreise werden die Pakete von den Routern nicht mehr verändert, sondern einfach nur weitergeleitet. Es gäbe keinen Sinn Operationen mit diese Paketen durchzuführen, da bereits Feedbackinformationen enthalten sind. Die Router belegen ausschließlich solche Pakete mit eigenen Informationen, wenn dies explizit gewünscht ist, da das PTP-Protokoll bis auf

zeitweises sammeln von Daten über die Verbindungsqualität, beispielsweise mit einem UDP-Protokoll kombiniert werden soll.

e) Benutzerschnittstelle

In Datei server.cc findet sich die Implementierung einer kleinen Benutzerschnittstelle, bei der über Menüpunkte die verschiedenen Fähigkeiten des Protokolls getestet werden können. Punkt 1 bis 4 beinhaltet die Abfragen von MTU und IF_SPEED jeweils mit verschiedenen Optionen. Bei Punkt 5 können diese Optionen vom Benutzer eingegeben werden. Für deren Hintergründe soll jedoch auf die Readme-Dateien oder aber auf das Buch zum PTP-Protokoll verwiesen sein.

Von Punkt 6 bis 8 konnte in der alten Implementierung der Trend getestet werden. Auf den Trend wird zum späteren Zeitpunkt noch einmal genauer eingegangen. In der aktuellen Implementierung wird mit Punkt 6 der Thread für die Messung gestartet und mit Punkt 8 gestoppt. Punkt 7 liefert die aktuellen Werte zurück, irrelevant ob während oder nach der Messung.

3.2 Unterschiede zwischen v3.0 und v5.0 im Detail

a) Version 3.0: Der Trend

Im PTP-Protokoll v3.0 war der Aussagegrad der Feedbackmessage nicht sehr hoch. Dem Server wurden lediglich Information darüber geliefert, ob der MIB2_IF_SPEED auf dem Pfad grösser oder kleiner wurde oder gleich blieb. Gemessen wurde mit einem Thread, welcher vom Benutzer über die Schnittstelle gestartet und gestoppt werden konnte. Dabei wurden PTP-Abfragepakete in einem gegebenen Intervall an den Client verschickt. Nach erfolgreicher Ankunft des zweiten Paketes konnte dieser den Trend berechnen und dem Server diese Information in einer Feedbackmessage mitteilen. Nach Ankunft jedes weiteren PTP-Paketes wurde der Trend neu berechnet und dem Server gesendet.

3.3 Version 5.0: Bandbreite und Traffic – das Feedbackpaket

Der Trend war für einen effizienten Einsatz nicht ausreichend. Die Daten werden auf dem Pfad ohnehin schon gesammelt und auch die Spezifikation wurde umgeschrieben. Deshalb musste auch die Implementierung umgeschrieben werden. Kern dieser Veränderung war die Feedbackmessage. Sie sollte nicht mehr nur den Trend, sondern nützliche Daten in einem spezifizierten Messageformat zurückliefern. Folgende Graphik zeigt die Struktur dieser Feedbackmessage:

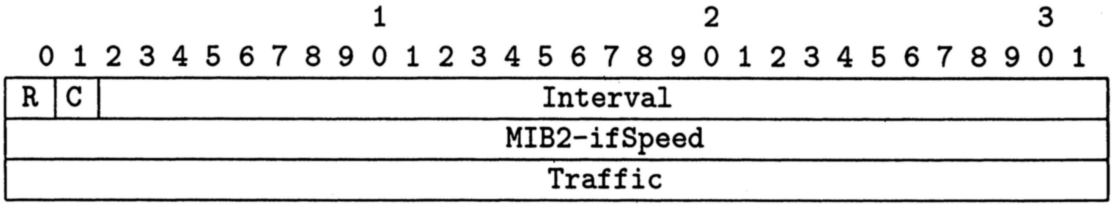


Abbildung 1: Available Bandwidth feedback message format

- Das erste Bit – falls gesetzt – signalisiert eine Änderung der Route, was bedeutet, dass der Testlauf nochmal gestartet werden muss.
- Das zweite Bit – falls gesetzt – signalisiert einen Counter Overflow, wobei in diesem Fall im darauffolgenden Feld (Interval) der maximale Wert für den Counter angegeben ist.
- Das Feld `Interval` gibt bei korrektem Ablauf der Messung das Interval an, in welchem die Messung stattgefunden hat.
- Das Feld `mib2_ifspeed` beinhaltet die kleinste Bandbreite auf der Route.
- Das Feld `traffic` beinhaltet den maximalen aktuellen PTP-Traffic auf der Route.

Diese Struktur ist in der Datei `ptpcommon.h` als `struct ptpmsg` definiert.

3.4 Änderungen an der Implementierung

a) Server

Die Änderungen am Server bestanden vorwiegend darin die vorhandene Implementierung des Threads, welcher auf Auswertung der Informationen des Clients wartet, so zu verändern, dass beim Senden der Auftragspakete der richtige ContentType verwendet wird und beim Entgegennehmen des Feedbackpaketes die Informationen richtig interpretiert und zwischengespeichert werden. Dies geschieht zwischen Zeile 477 und 586 in Datei `ptpserver.cc`.

Zum Zwischenspeichern der ausgewerteteten Informationen wurden zur Klasse `PTPserver` einfachheitshalber dieselben Variablen hinzugefügt die bereits im Antwortpaket enthalten sind. Weiters gibt es eine zusätzliche Funktion namens `get_bandwidth`, welche die obengenannten Werte ausliest und an den Aufrufer zurückgibt.

b) Client

Auch am Client konnte der bereits bestehende Code in etwas veränderter Form wiederverwendet werden. Ähnlich wie beim Server gibt es auch hier einen Thread der auf ankommende Pakete wartet, diese nach ihrem Inhalt auswertet und je nachdem die zugehörige Funktion aufruft. So wird im Fall der Bandbreitenmessung die Funktion namens `process_addr_time_ifspeed_ifoctets(packet, len);` aufgerufen, wobei `paket` der Zeiger auf das angekommene Paket und `len` dessen Länge ist.

Da die Funktion zum Erstellen und Updaten der Adresstabelle, in welchem die Werte der Router in einer Liste abgelegt werden, schon existierte, musste diese nur mit den richtigen Parametern aufgerufen werden. Um diese Informationen auszuwerten und brauchbare Informationen zum Server zurückzuschicken, musste die anschließend aufgerufene Funktion

`handle_bandwidth_measurement(...);` neu implementiert werden.

Diese Funktion ist in 3 große Bereiche unterteilt, da das Feedbackpaket einen Route-Change-Error, einen Counter-Overflow-Error oder die Feedbackinformationen zurückliefern kann. Insofern wird gleich am Beginn geprüft ob es eine Routenänderung gegeben hat und ev. angemessen darauf reagiert. War dies nicht der Fall, wird die Adresstabelle aktualisiert. Scheinen die Informationen in jedem Datensatz korrekt zu sein, erfolgt die Berechnung der minimalen Bandbreite und des Traffics. Bevor die angeforderten Informationen verpackt und verschickt werden können wird noch untersucht ob es einen Counter-Overflow gab. War dies der Fall wird auch hier angemessen reagiert und im Paket die Signalisierung des Fehlers sowie der maximal zulässige Wert mitgeliefert. Waren bisher alle Fehlerprüfungen negativ, so werden die Flags für die Fehlersignalisierung auf 0 gesetzt und die Feedbackinformationen dem Server geschickt.

c) Router

Zwar wurde sowohl am Client- wie auch am Servercode relativ viel verändert, trotzdem waren am Router keine Modifikationen notwendig. Die bestehende Implementierung beinhaltet bereits das Sammeln aller Daten, sodass am Router (= Kernelpatch) nur eine einzige Codezeile angepasst werden musste. Jene in welcher der Traffic am Netzwerkinterface gemessen wird. Dort wurden sowohl ein- als auch ausgehende Bytes gemessen, was das Trafficvolumen verdoppelt. In der neuen, aktuellen Implementierung sind nur mehr die ausgehenden Bytes relevant.

d) Benutzerschnittstelle

An der Benutzerschnittstelle wurde nicht wesentlich viel verändert, lediglich so angepasst, dass anstatt dem Trend die neuen Information ausgegeben werden können. Dies gelingt durch Aufruf der oben besprochenen Funktion `get_bandwidth()` aus der Klasse `PTPserver`.

4. Testen von PTP v5.0

4.1 Vorgangsweise und Probleme

Das Testen der neuen Implementierung unterschied sich nicht vom Testen der Alten. Verwendet wird hierzu das Netzwerktool Ethereal bei dem die PTP-Pakete auf jedem Rechner visuell verfolgt und gleichzeitig dessen Inhalt überprüft werden können. So wurde der oben genannte Fehler gefunden und zwar, dass der gemessene Traffic immer doppelt so hoch war wie er eigentlich sein sollte.

5. Die Zukunft von PTP

5.1 Anpassung auf einen aktuellen 2.6-er Kernel

Nachdem der Kernel 2.6 langsam aber sicher zum Standard wird, liegt es nahe den PTP-Kernelpatch auch für diesen Kernel anzupassen. Auch hier wird es notwendig sein die Netzwerkimplementierung sehr genau auf die Änderungen zu untersuchen und anschließend Schritt für Schritt den Kernel manuell zu patchen. Eine genaue Aufwandschätzung kommt dem Aufwand der Anpassung an den Kernel ziemlich gleich, sodass dies wohl im selben Zug gemacht werden wird. Es dürfte es wohl nicht viel umfangreicher sein als der Schritt vom Kernel 2.2 auf Kernel 2.4. Diesem Projekt kommt die Erfahrung mit Protokoll und Netzwerkimplementierung des Linuxkernels zugute kommen.

5.2 Emulab als Simulation der Wirklichkeit

Emulab ist eine Forschungseinrichtung bei der weit über hundert Computer vernetzt sind um Netzwerke zu simulieren. Der Benutzer, der ein solches Projekt startet, kann die Verbindungen und Netzwerkknoten beliebig konfigurieren und somit ein komplette Netzwerkszenario aufbauen.

Dies ist ideal um brauchbare Ergebnisse des PTP-Protokolls in einem

wirklichkeitsnahen Netzwerkszenario zu erhalten, da der Einsatz des Protokolls in der Wirklichkeit oder wenigstens in einer wirklichkeitsnahen Umgebung das gesamte PTP-Projekt weiterbringen würde. Dies kann im Zuge einer weiteren Bakkalaureatsarbeit durchgeführt werden.

5.3 Implementierung als eigenständiges Protokoll

Ziel des PTP-Protokolls ist es, in Verbindung mit z.B. CADPC, als eigenständiges Protokoll beispielsweise im Teilnetz eines Providers eingesetzt zu werden. Hierzu müsste allerdings jeder Router im Teilnetz des Providers ein PTP-Router sein. Schon die Ausnahme eines einzigen Routers der kein PTP versteht, kann den Vorteil des Protokolls im gesamten Netz nichtig machen (sofern er nicht die schnellste Verbindung aller Router im Netz hat).

Denkbar wäre der Einsatz in einer kleineren, überschaubareren Umgebung, wo sich PTP, aufgrund des geringeren Administrationsaufwands, leichter durchsetzen könnte.

6. Literaturverzeichnis

- Michael Welzl.:Scalable Performance Signalling And Congestion Avoidance. Kluwer Academic Publishers, August 2003, ISBN 1-4020-7570-7
- Alessandro Rubini, Jonathan Corbet: Linux Device Drivers, 2nd Edition. O'Reilly Juni 2001, ISBN 0-596-00008-1
- The Linux Kernel API, GNU