INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenen Praktikum

Aufbau eine MPLS Testbeds für Linux

Patrick Havel

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Harald Rölle

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Das schnelle Wachstum und die steigenden Anforderungen an die Dienstqualität sorgen für eine ständige Veränderung im Internet. Es findet eine dauernde Anpassung an die jeweiligen Bedürfnisse statt.
Eine dieser neueren Anpassungen ist Multi Protocol Label Switching (MPLS). Dies ist eine durch die Internet Engineering Task Force (IETF) spezifizierte Protokollfamilie. MPLS wird in [RVC01] beschrieben. Dieser zum Standard gereifte RFC beschreibt sowohl die Bestandteile als auch mögliche Anwendungen. Desweiteren findet eine Spezifikation der Vorgänge innerhalb eines MPLS Netzwerkes statt.

1.1 Aufgabenstellung

Das Ziel dieses Fortgeschrittenen Praktikums ist die Realisierung eines Rechnernetzes das `Multi Protocol Label Switching (MPLS)` unterstützt. Eine Einführung in MPLS wird im nächsten Abschnitt gegeben.
Diese neuartige Technologie soll innerhalb eines überschaubaren Testbed etabliert werden. Durch den Aufbau eines solchen Testbeds entstehen vielfältige Möglichkeiten um Tests mit `MPLS` durchzuführen.
Zur Realisierung des Testbeds sollen einige Rechner innerhalb eines größeren Netzes mit MPLS versehen werden. Die entstandene Testumgebung soll auch von den parallel laufenden `Voice over IP` Fortgeschrittenen Praktika genutzt werden können.

1.2 Einführung in MPLS

Das Internet arbeitet als verbindungsloses Netzwerk. Das bedeutet es werden Pakete von Router zu Router verschickt, wobei jeder dieser Router eine unabhängige Forwarding Entscheidung trifft. Diese basiert einerseits auf dem verwendeten Routing Algorithmus und andererseits auf dem Inhalt

des Paketheaders des zu verschickenden Paketes. In der Regel enthält so ein Paketheader mehr Informationen als für die Next-Hop Bestimmung notwendig wären. Man kann die Bestimmung des nächsten Router auch als zweistufige Aktion betrachten. Zunächst werden alle Pakete die das gleiche Ziel haben in sogenannte Forwarding Equivalence Classes (FEC) eingeteilt um dann im nächsten Schritt alle möglichen Folgerouter auf diese FECs abzubilden.
Beim konventionellen IP Forwarding wie es im Internet heutzutage eingesetzt wird, wird ein Router zwei Pakete als der gleichen FEC angehörig betrachten, falls diese eine gemeinsame Zieladresse haben, bzw. einen hinreichend großen gemeinsamen Adressanteil. Auf dem Weg durch ein Netzwerk wird das Paket an jedem Router erneut untersucht und einer FEC zugeordnet. Es muß also an jedem Router immer wieder die gesamte Forwarding Tabelle untersucht werden, um den nächsten Router für ein Paket zu bestimmen. Man spricht in diesem Zusammenhang auch von Next-Hops.
Bei MPLS findet diese Zuordnung Paket zu FEC nur einmalig statt und zwar am Anfang des MPLS Netzwerkes am sogenannten `Ingress Router`. Pakete die der gleichen FEC angehören sind bei MPLS nicht zu unterscheiden. Alle Pakete die einer FEC angehören und vom gleichen Router starten werden also den gleichen Weg nehmen. Diese im Netzwerk etablierten Wege nennt man bei MPLS Label Switched Path (LSP). LSPs sind unidirektionale Wege zum Datentransfer, die die Pakete einer FEC transportieren. Im Gegensatz zum verbindungslos arbeitendem IP wird bei MPLS also eine verbindungsorientierte Kommunikation verwendet. Ein LSP beginnt an einem Ingress Router der dem zu transportierenden Paket ein Label auf den Label Stack schiebt. Auf diese wird im weiteren noch eingegangen.
Ein MPLS Netzwerk bildete eine Domäne deren Grenzrouter eingangsseitig als Ingress Router und ausgangsseitig als Egress Router bezeichnet werden. Unter einer Domäne wollen wir laut Standard eine `Menge von benachbarten Knoten die MPLS Forwarding und Routing verwenden und sich ebenfalls in einem Verwaltungsbereich befinden` verstehen. Router die entlang eines LSP liegen bezeichnet man als Label Switch Router (LSR), solche die die Endpunkte bilden als Label Edge Router (LER).
Alle LSRs treffen anhand des jeweiligen Labels ihre Forwarding Entscheidung. Hier liegt auch der große Unterschied zu IP, da durch die Beförderung entlang eines LSP keine lokal bestimmte Wegewahl mehr statt findet, d.h. das die Router keine aktive Next Hop Entscheidung treffen können wie sie bei IP üblich ist. Der LSP endet am Egress Router der den Label Stack auf eine Tiefe kleiner der am Ingress Router reduziert. Das Paket kann vom Egress Router noch weiter verschickt werden, was für uns allerdings uninteressant ist, da es die MPLS Domäne oder ein Subnetz verlassen hat.

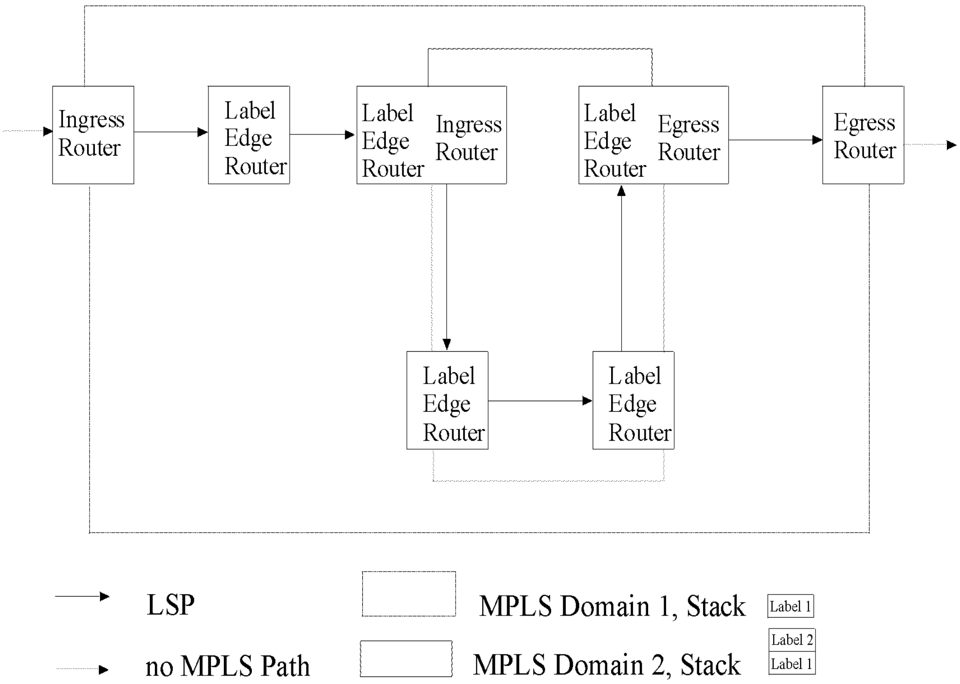

Abbildung 1.1: MPLS Domäne mit Labelstacktiefe 2

Label sind Informationseinheiten die dem bestehenden Header vorangestellt werden. Es handelt sich aber nicht um eine Kodierung der Header Adresse. Vielmehr bestimmt dieses Label zu welcher FEC ein Paket gehört. Beim Forwarding wird an einem Router nun der Wert des jeweiligen Labels betrachtet um zu einer Forwarding Entscheidung zu gelangen. Dieser Wert wird dann als Index in der Forwarding Table des jeweiligen Routers verwendet.

Ein Paket kann mehrere Labels vorangestellt bekommen. Diese werden im sogenannten Label Stack verwaltet. Diese nach dem First In First Out (FIFO) Prinzip verwaltete Strukturierung der Labels ermöglicht eine Labelung von Paketen für Subnetze. Bewegt sich ein Paket zum Beispiel in einem Netze ohne Subnetze so hat der Label Stack die Tiefe eins, vgl. auch [RTF+01].

Der Multiprotokoll Aspekt bei MPLS geht darauf zurück, das es für MPLS unerheblich ist, welches Network Layer Protocol zugrunde liegt. Es können sowohl Frame Relay [BBM92], ATM AAL 5 [Atk94] als auch Ethernet [Plu82] verwendet werden.

Kapitel 2

Produktauswahl und Kriterienkatalog

In den folgenden Abschnitten wird zunächst kurz auf die Produktauswahl eingegangen. Danach werden die gefundenen Produkte vorgestellt. Danach wird der Kriterienkatalog hergeleitet und darauf basierend das Siegerprodukt präsentiert.

2.1 Produktauswahl

Auf Grund der beschränkten Zeit und um den Aufwand im Rahmen zu halten, fand eine einfach gehaltene Analyse der Produkte statt. Konkret wurden die im folgenden Abschnitt gelisteten Produkte aufgrund ihrer Beschreibung mit dem Kriterienkatalog verglichen. Es fand also keineswegs eine Installation aller im Internet gefundener Produkte statt.

2.2 Kriterienkatalog

Die zu verwendende Implementierung sollte einige Anforderungen erfüllen, um ein gut funktionierendes Testbed aufzubauen. Diese werden im folgenden Kriterienkatalog genannt und begründet.
Das Betriebssystem sollte sowohl aus Kostengründen, als auch aufgrund der Möglichkeit direkt im Betriebsystem Einstellungen vorzunehmen Linux sein. Eine OpenSource Implementierung war gewünscht um die Möglichkeit zu haben, ggf. im SourceCode Änderungen vorzunehmen.
Eine der wichtigsten Anforderungen an das Testbed war eine möglichst hohe Konformität mit dem Standard. Einige der wichtigsten Funktionalitäten sind z.B. das `Label Distribution Protocol (LDP)`, die `Label Switched Paths (LSP)` und natürlich der `Label Stack`. Die Spezfikation der genannten Techniken kann in [DLM+01], [ADF+01], [TG01] und [NKD+01] nachgelesen werden.

Die Implementierung sollte über eine möglichst gute Dokumentation verfügen. Die Existenz einer solchen erleichtert die Arbeit sowohl für die Erstellung des Testbeds als auch für weitere User. Ein weiteres Kriterium war der Wunsch nach Support in irgendeiner Form. Ob dieser über Newsgroups, eine Mailingliste oder ganz anders stattfinden sollte, spielte dabei keine Rolle.
Die Form der Implementierungstechnik wurde ebenfalls begutachtet, da eine Einbindung in das Testbed ausschließlich softwareseitig realisiert werden konnte. Da außerdem nach einer `OpenSource` Implementierung gesucht wird, ist eine Begutachtung der internen Realisierung ebenfalls möglich.
Die Kontinuität der Implementierung bzw. ihrer Entwicklung war ebenfalls ein wichtiger Punkt, da das realisierte Testbed ggf. auch für weitere Untersuchungen zur Verfügung stehen sollte.
Während der Recherche im Internet wurden drei mögliche Implementierungen gefunden. Diese werden im folgenden vorgestellt und anhand des gerade erstellten Kriterienkatalogs klassifiziert.

2.3 Die Implementierungen

2.3.1 Nortel Networks

Bei der Implementierung von Nortel Networks handelt es sich um eine C Library. Sie ist zwar sowohl ein Open Source Produkt, als auch für Linux erhältlich, allerdings ist die Standardkonfomität bezüglich unserer Anforderungen keineswegs sichergestellt, z.B. ist der `Label Stack`, eine zentrale Komponente von MPLS, überhaupt nicht implementiert.
Die beiliegende Dokumentation ist eher notdürftig bzw. als Gedächtnisstütze für den Programmierer zu sehen.
Es wird kein Support von Nortel Networks bezüglich dieser MPLS Implementierung angeboten. Es gibt auch keine Newsgroup oder FAQ Seite zu diesem Thema auf der entsprechenden Webseite.
Die bisher wohl am besten als rudimentär charakterisierte Implementierung wird allerding vorangetrieben. Generell kann man sagen das diese Implementierung noch ein wenig Zeit braucht, um ein Testbed wie es hier vorgestellt wird, damit zu betreiben.

2.3.2 Cell-Relay

Die Cell-Relay Implementierung von MPLS wurde von Keir Fraser und Phil Quiney realisiert.
Sie arbeitet als Kernel Modul und wird über ein beigefügtes Tool mit Namen `mpls` gesteuert.
Die Implementierung ist sowohl ein Open Source Produkt als auch für Linux verfügbar.

In dem heruntergeladenen Archiv ist eine ausführliche Dokumentation enthalten, die das Arbeiten mit dem Produkt leicht gestaltet. Leider wird von Cell Relay keinerlei Support angeboten und auch hier existieren weder Newsgroups noch aktualisierte FAQ Seiten.
Die größte Schwäche dieser Implementierung ist jedoch die mangelhafte Standardkonformität. So wird LDP überhaupt nicht und der Label Stack nur im eingeschränkten Maße unterstützt.
Anhand der Webseite läßt sich nicht erkennen, ob und wie dieses Projekt weiterverfolgt wird.

2.3.3 SourceForge

Die MPLS-Implementierung von James R. Leu, die auf `www.sourceforge.org` frei verfügbar zum Download bereitsteht, ist als Kernel Patch für den Linux Kernel, Version 2.4.1, realisiert.
Es gibt eine Mailingliste auf `www.sourceforge.org`, die sich ausschließlich mit Fragen zu dieser Implementierung beschäftigt, so daß ein Support gesichert ist.
Eine Anleitung zur Installation und eine umfangreiche Dokumentation zur Benutzung des Tools `mplsadm` sind in dem Paket enthalten.
Die Kontinuität der Implementierung scheint z.B. durch die Angabe der noch zu implementierenden Teile von MPLS, gewährleistet.
Der Download und die Installation der Implementierung erwiesen sich als problemlos.
Mittels der mitgelieferten Dokumentation können manuell LSPs etabliert werden. Nach der Etablierung dieser kann man im /proc Filesystem entsprechende Einträge finden.

2.3.4 Zusammenfassung

Hier noch einmal ein tabellarische Übersicht über erstellten Kriterienkatalog und die einzelnen Produkte.

Wie aus der Tabelle ersichtlich, entspricht die Implementierung von Source-Forge exakt dem Kriterienkatalog. Das im folgenden Abschnitt beschriebene Testbed wurde folglich mit dieser Implementierung realisiert.

Die Implementierungen

Kriterien \ Produkt	SourceForge	Cell Relay	Nortel Networks
Linux	Ja	Ja	Ja
Open Source	Ja	Ja	Ja
Standardkonformität			
LDP	Ja	Nein	Ja
LSP	Ja	Ja	Ja
Labelstack	Ja	eingeschränkt	Nein
Dokumentation	Ja	Ja	Nein
Support	Mailingliste	Nein	Nein
Implementierungstechnik	Kernel−Patch	Kernel Modul	C Library
Kontinuität	Ja	???	Ja

Abbildung 2.1: Die Implementierungen

Kapitel 3

Das Testbed

In den folgenden Abschnitten wird zunächst der Aufbau des Testbeds beschrieben. Es wird zunächst auf die verwendete Hardware eingegangen und anschließend das Vorgehen für die Installation der Software beschrieben. Der darauffolgende Abschnitt zeigt die verwendete Topologie auf und erläutert deren Zweckmässigkeit. Es folgt die Verifikation das MPLS tatschlich verwendet wird und es wird auf Probleme der verwendeten Implementierung eingegangen. Den Abschluß bildet ein ausführliches Beispiel.

3.1 Erstellung

Der Aufbau des Testbeds beginnt zunächst mit dem Aufstellen der einzelnen Rechner. Es soll die 100 Mbit/s Variante von Ethernet (`FastEthernet`) verwendet. Zur Verbindung der einzelnen Karten kommen `Twisted Pair` Kabel zum Einsatz.
Alle Rechner wurden mit einer für die Topologie (vgl. auch nächsten Abschnitt) notwendigen Anzahl an Ethernet Karten versehen. Bei diesen Karten handelt es sich um handelsübliche Netzwerkkarten, unter anderem der Firma 3Com.
Nach dem Aufstellen der Maschinen erfolgt eine Vernetzung der einzelnen Maschinen mittels Cross Link Kabeln.
Um eine bessere Übersichtlichkeit des Testbeds zu erreichen wurden Subnetze der Klasse C zwischen den einzelnen Rechnern etabliert. Die von Linux hierfür verwendeten boot Skripte wurden dementsprechend angepasst bzw. erweitert.
Aus Zeitgründen wurde auf die Verwendung eines `DNS-Servers` verzichtet. Alle Maschinen wurden aber mittels Einträgen in der Datei `/etc/hosts` mit symbolischen Namen versehen. Dies macht ein adressieren der Rechner für den Menschen leichter, da nicht mehr die IP Adresse verwendet werden muß. Vielmehr werden statt dessen symbolische Namen verwendet, die bijektiv auf die IP Adresse abgebildet werden. Eine solche Datei ist im

Anhang gelistet.
Um nicht bei jedem booten das gewünschte Routing manuell zu erzeugen, wurden lokale Skripte, die beim Booten von Linux gestartet werden entsprechend angepasst. Konkret handelt es sich hierbei um die Datei `boot.local`. In dieser Datei wurde auch ein Nachladen eventuell fehlender Treiber angestossen. Dies musste für jeden Rechner individuell geschehen, da Netzwerkarten verschiedener Hersteller zum Einsatz kommen. Eine solche Datei ist im Anhang gelistet.
Die Korrektheit der Verbindungen wurde mittels ping zwischen den einzelnen Maschinen verifiziert.
Die durch den Kriterienkatalog spezifizierte Implementierung von `sourceforge` verlangt nach der Kernel Version 2.4.1. Dieser ist nicht Bestandteil der verwendeten Linux Distribution, wie sie auf den einzelnen Rechnern zu diesem Zeitpunkt installiert ist. Folglich mußte auf den vier `mplsX` Rechnern dieser Kernel installiert werden. Die Sourcen die für die Kompilierung eines Kernel benötigt werden, können auf `www.kernel.org` heruntergeladen werden. Nach dem Entpacken der Sourcen wurde der Kernel mit den benötigten Optionen kompiliert. Da es sich um verschiedene Rechnertypen handelt, ist eine individuelle Anpassung für das Testbed notwendig. Eine Modifikation des `Linux Loaders (LiLo)` stellt sicher das der neue Kernel per default gebootet wird. Die Kernelversion kann durch Eingabe von `uname -r` verifiziert werden.
Da die gewählte MPLS Implementierung als Kernel Patch realisiert ist, muß dieses noch installiert werden. Nach dem Entpacken und Installieren des Patches ergeben sich zusätzliche Kerneloptionen.
Nach Wahl der Optionen `Kernel/User Netlink Socket`, `Routing Messages` und `Multi-Protocol Label Switching` wird der Kernel neu kompiliert und das alte Image mit dem neuen überschrieben.

Nach der manuellen Etablierung eines LSPs (siehe `Einführung in MPLS`) zu Testzwecken, kann in `/proc/net/mpls*` die gewählte Konfiguration überprüft werden. Es handelt sich bei `/proc/*` um ein virtuelles Dateisystem das interne Kernelstrukturen darstellt.
Das während des FoPras erstellte Testbed besteht aus sechs Maschinen, deren Topologie und Nutzung im folgenden Abschnitt erläutert werden.

3.2 Topologischer Überblick und Benutzung

Die Abbildung 4 zeigt die Topologie des gesamten Testbeds.

Die Wahl dieser konkreten Topologie hatte verschiedene Gründe. Zum

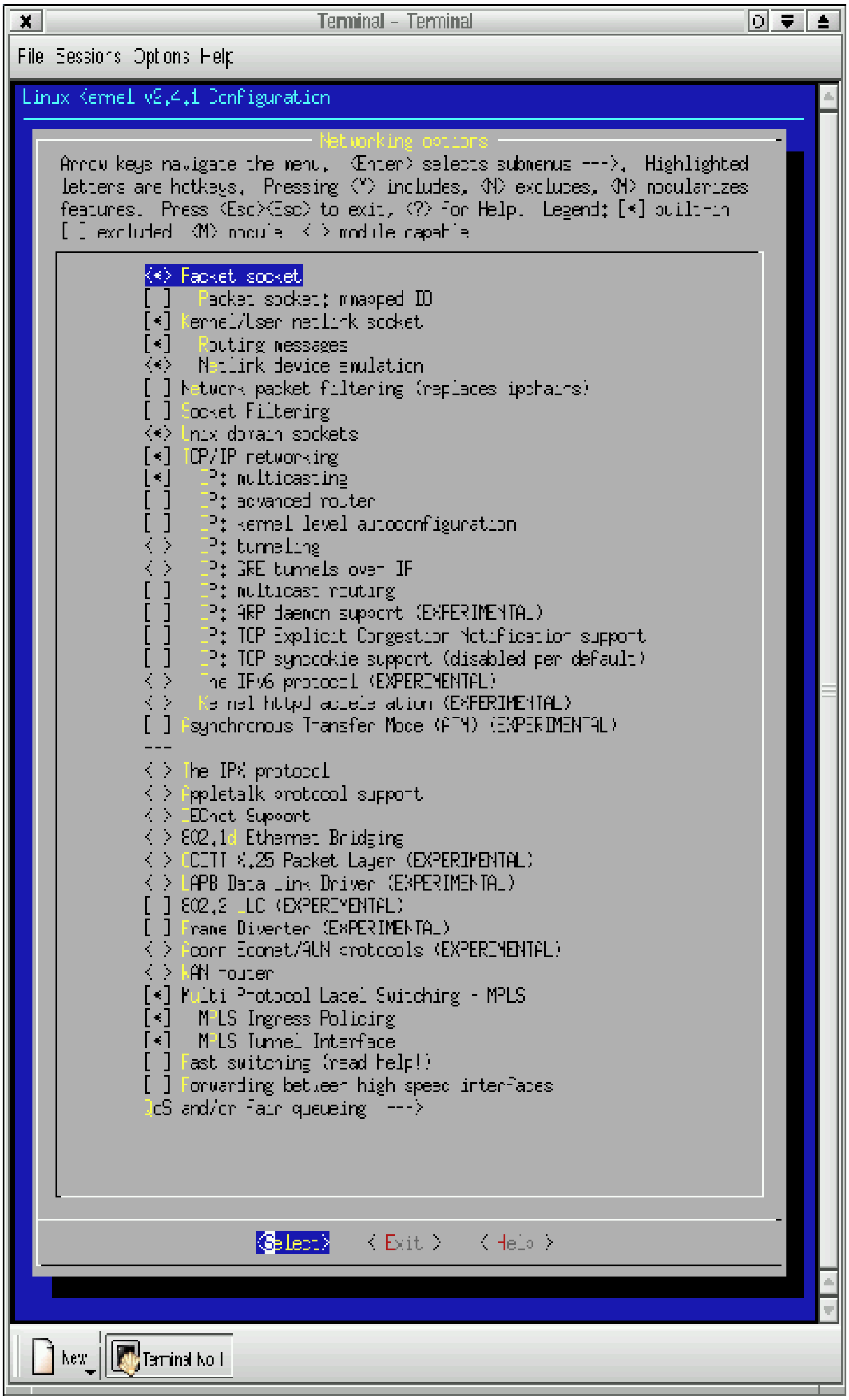

Abbildung 3.1: Screenshot der gewählten Kerneloptionen

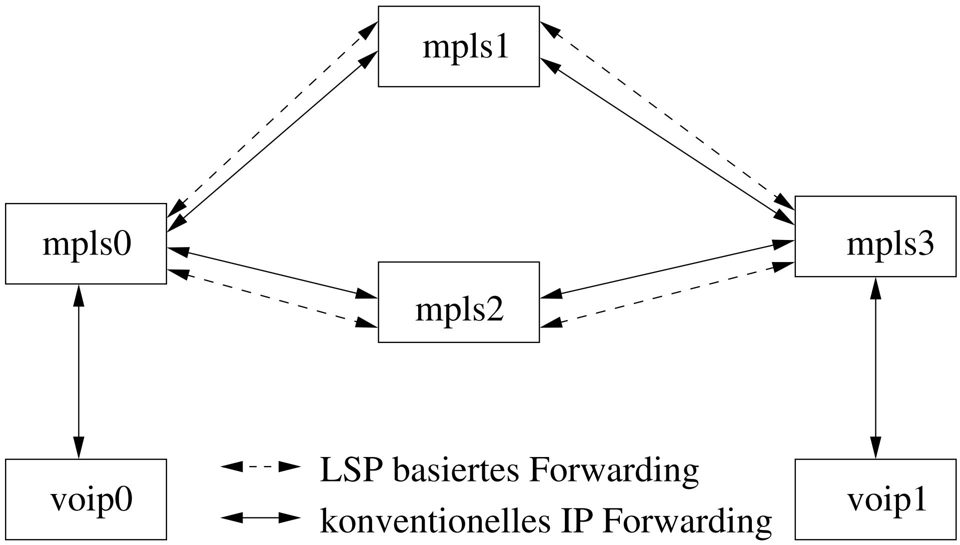

Abbildung 3.2: Topologischer Überblick

einen gestattet sie die Messung auf zwei parallel laufenden Strecken, also mpls0-mpls1-mpls3 und mpls0-mpls2-mpls3. Zum anderen können längere LSPs durch ein "im Kreis laufen "simluliert werden. Dies ist möglich da man mittels MPLS an jedem `Label Edge Router(LER)`, in diesem Fall also alle `mplsX` Rechner, ein neues Label vergeben darf und so tatsächlich bei jedem Durchlauf einen anderen LSP verwenden kann.
Die vier mit `mplsX` gekennzeichneten Maschinen besitzen alle den Linux Kernel 2.4.1, inklusive dem SourceForge MPLS Patch, vgl. auch vorigen Abschnitt. Die beiden voipX Maschinen arbeiten mit dem Kernel 2.2.16. Diese Konfiguration kommt daher das die voipX Rechner im Gegensatz zu den mplsX Rechnern keinen speziellen Kernel benötigen.
Das Gesamte Netz ist vollständig durchgeroutet. Zusätzlich ist die Maschine voip1 an das Internet angeschlossen und dient als Maskerading Server. Dies ermöglicht einen `remote` Zugang von außen, zum Beispiel mittels `Secure Shell (ssh)`.
Parallel zum Aufbau des Testbeds laufen noch einige `Voice over IP` FoPras. Diese haben durch die Verwendung von Skripten die Möglichkeit eine MPLS Teststrecke zu etablieren.
Die gerade erwähnten Skripte etablieren vordefinierte LSPs im Testbed. Sie werden zum Beispiel mit `gompls_mpls1` gestartet. Um einen Vergleich der Meßstrecke mit herkömmlichem IP Forwarding zu bekommen gibt es auch Säuberungsskripte die die LSPs wieder zerstören. Ein Aufruf erfolgt beispielsweise mittels `gompls_mpls1_clean`. Ein beispielhaftes Listing findet sich im Anhang. Die folgende Abbildung zeigt die Situation zwischen den

Rechnern mpls0 und mpls1 nach Ausführung des Skripts `gompls_mpls1` auf mpls0 bzw. `gompls_mpls0` auf mpls1.
Zur Verifikation von MPLS auf den Maschinen wurde ein Softwarebasierter Protokollanalysator verwendet. Im konkreten Fall handelt sich hierbei um `Ethereal`. Die folgende Abbildung zeigt den MPLS Header wie er im Standard vorgesehen ist und einen Screenshot von Ethereal. Die Pfeile geben an, wo im gemessenen ping Verkehr welcher Teil des Headers zu finden ist.

3.3 Probleme der verwendeten Implementierung

Das einzige Problem das diese Implemntierung gegenüber unserem Kriterienkatalog hat, ist die mangelnde `Label Distribution Protocol (LDP)` Funktionalität. Diese ist rudimentär als User Space Library implementiert. Die beigefügten Beispiele konnten jedoch nicht zum laufen gebracht werden, so daß für diesen Punkt lediglich die Möglichkeit bleibt, eine neue Version abzuwarten.
Eine diesbezügliche Hoffnung stellte das `Zebra`-Projekt dar. Es gibt eine Ankündigung auf `www.sourceforge.net`, das eine Integration des LDP mit der `Zebra Routing Plattform` stattfinden soll.

3.4 Ausführliches Beispiel

In diesem Abschnitt Etablierung zweier LSPs mittels der `Sourceforge`beschrieben werden.
Die allgemeine Syntax des Tools `mplsadm` auf Kommandozeilen Ebene ist wie folgt:

- mplsadm [ADBUdhvT:f:L:I:O:i:o:]

  A add modifier

  B bind modifier

  D delete modifier

  U unbind modifier

  d toggle debug

  h this message

  v verbose info

  T tunnel name:dest addr

  f prefix/len

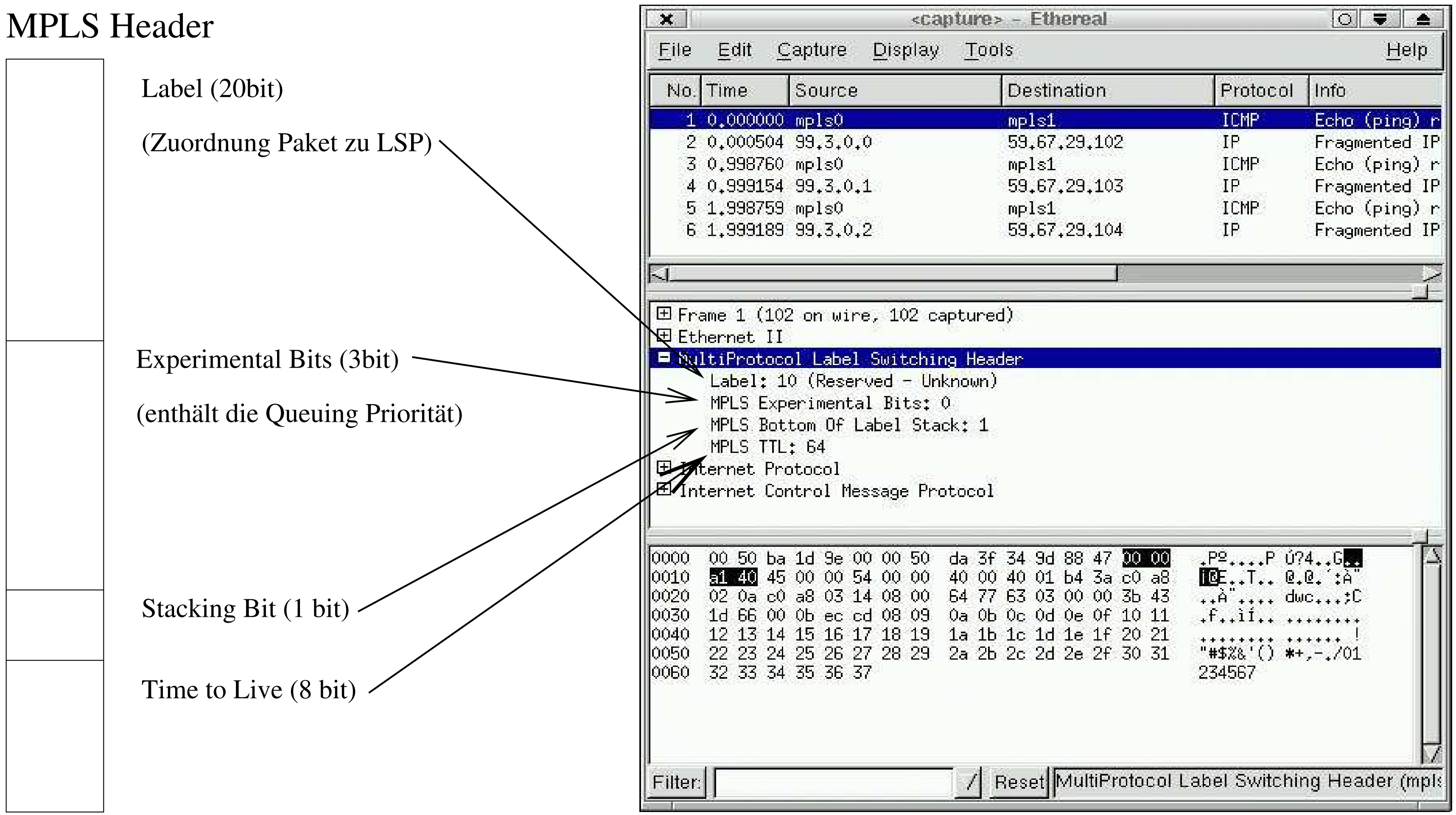

Abbildung 3.3: Screenshot von Ethereal mit MPLS Header

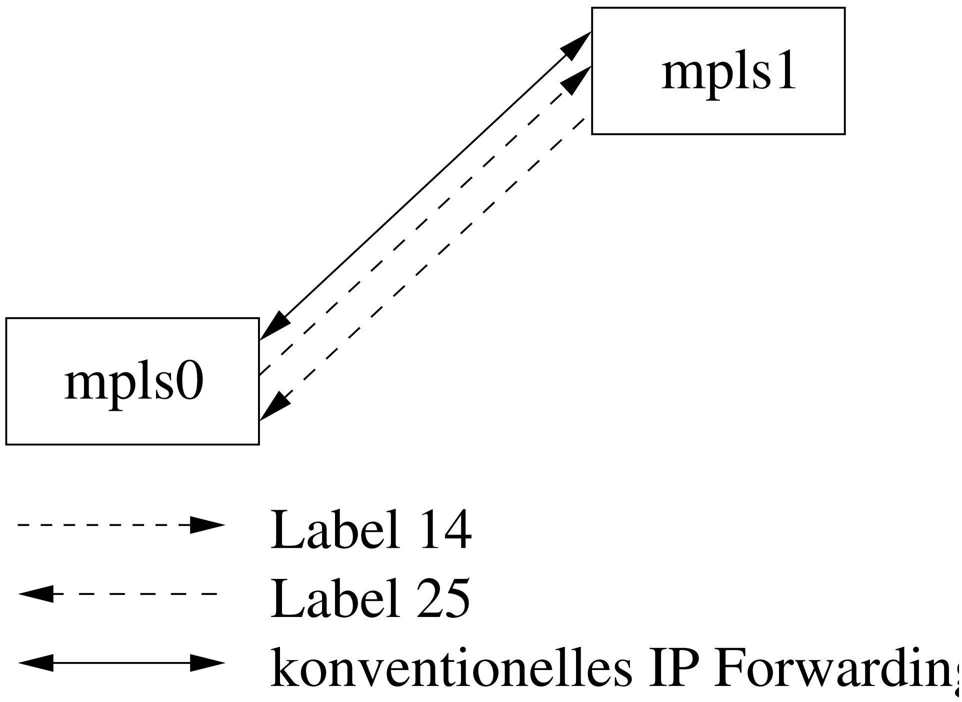

Abbildung 3.4: Ausschnitt des LSPs im Testbed

L interface name:label space, set the label space for an interface (-1 disables)

I gen—atm— fr:label:label space create—delete an incoming label

O gen—atm—fr:label:out interface[:next type:next hop]

i opcode:opcode_data

o opcode:opcode_data

Im folgenden wird die Eingabe auf der Maschine `mpls0` betrachtet um die in der folgenden Abbildung dargestellte Situation zu erreichen.

Es wird immer zunächst die benötigte Eingabe betrachtet und dann der Inhalt der `proc/net/mpls*` zur möglichen Verifikation aufgezeigt.

```
mplsadm -A -B -O gen:14:eth0:ipv4:192.168.2.20 -f
192.168.3.20/32
```

Diese Zeile hat einen auswärtsgerichteten (-O) LSP zum Rechner hinzugefügt (-A) und gebunden (-B). Es wird das Label 14 auf dem Interface eth0 verwendet, wobei 192.168.2.20 die IP Adresse des Empfängers darstellt.

Der letzte Teil spezifiziert die zu verwendende FEC.
in mpls_out: `40002802 PUSH(gen 14) SET(eth0)`
in mpls_fec: `40002802 192.168.3.20/32`

`mplsadm -L eth0:0`

Die Schnittstelle eth0 soll den Labelspace 0 verwenden
in mpls_labelspace: `eth0 0`
`mplsadm -A -I gen:25:0`
Ankommende Labels (25) werden im Labelspace 0 abgelegt.
in mpls_in: `40005000 gen 25 0 POP DLV`

Kapitel 4

Zusammenfassung und Ausblick

MPLS stellt einen Versuch dar dem verbindungslosen Internet eine Verbindungsorientierung aufzuprägen. Ein ganz ähnlicher Versuch fand schon mittels des `Asynchronous Transfer Mode (ATM)` statt, vgl. auch [LH98]. Man wird sehen ob und wenn welche Technik sich durchsetzen kann.
Die im ersten Teil gegebene Einführung stellt MPLS in groben Zügen dar. Im folgenden wurde in der Realität nach einer Implementierung gesucht. Zur Realisierung sollte die gesuchte Implementierung allerdings einige Kriterien erfüllen die im Kriterienkatalog hergeleitet und begründet wurden. Aus dem Kriterienkatalog und der Einsicht in die Funktionalität der Kriterien ergab sich die Implementierung von `www.sourceforge.net` als Sieger.
Nach dem Aufstellen der Maschinen und einer Etablierung der Randbedingungen, wie z.B. Topologie festlegen, Subnetze definieren, Boot Skripte anpassen, Kernel installieren, konnte mit der Installation der eigentlichen MPLS Software begonnen werden.
Als nächstes fand eine Verifikation der jeweiligen Sachverhalte statt und es wurde ein grundlegendes Problem im aktuellen Stadium der Implementierung gefunden.
Die zu Erwartende Kontinuität der Implementierung und die Existenz des `Zebra`-Projekts lassen aber auf eine baldige Behebung dieses Problems hoffen.
Das bisher erarbeitete Testbed ist keinesfalls ausgereizt. Es stellt vielmehr eine Grundlage für eine Reihe weiterer Versuche dar. Im folgenden sollen einige Vorschläge diesbezüglich formuliert werden.
Es wurden bisher beispielsweise keinerlei `Network Layer Controls` installiert oder getestet. Desweiteren sind in der jetztigen Ausbaustufe auch noch keine `Quality of Service Erweiterungen` installiert. Gerade dieser Aspekt von MPLS wäre durchaus einer nähreren Betrachtung wert. MPLS versucht ja gerade durch die Verbindungsorientierheit solche Dienst-

qualitätszusicherungen zu machen. Auch ein Test mit anderen `Schicht 2` Protokollen wäre durchaus denkbar und sinnvoll.
Desweiteren kann das Testbed für weitere Tests der mehrfach erwähnten `Voice over IP` FoPras verwendet werden.
Man sieht deutlich das noch ein großes Potential enzüglich weiterer Verwendung in diesem Testbed steckt.

Kapitel 5

Anhänge

Im folgenden werden die erstellten Skripte die zur Erstellung bzw. Vernichtung der einzelnen LSPs auf den mplsX Maschinen eingesetzt wurden aufgelistet.

5.1 Anhang A: Etablierung eines LSPs

Etablierung des LSPs von mpls0 zu mpls1

```
#! /bin/bash
#LSP von mpls0 nach mpls1 via mpls2
route add -host 192.168.3.20 gw 192.168.2.20
mplsadm -A -B -O gen:10:eth0:ipv4:192.168.2.20 -f 192.168.3.2
#LSP von mpls1 nach mpls0 via mpls2
mplsadm -L eth0:0
mplsadm -A -I gen:20:0
```

5.2 Anhang B: Zerstörung des LSPs

```
 #! /bin/bash
#aufräumen
route del -host 192.168.3.20 gw 192.168.2.20
mplsadm -D -B -O gen:10:eth0:ipv4:192.168.2.20 -f
192.168.3.2mplsadm -D -I gen:20:0
ifconfig eth0 down
ifconfig eth1 down
ifconfig eth2 down
```

5.3 Anhang C: Ein Bootskript

Das beschriebene Skript ist unter `/etc/init.d/boot.local` zu finden. Die Datei `boot.local` stellt eine saubere Initialisierung des Rechners bezüglich

Netzwerkkarten sicher.

```
. /etc/rc.config
modprobe via-rhine
/sbin/depmod -a
/sbin/modprobe via-rhine
/sbin/ifconfig eth0 192.168.2.10 netmask 255.255.255.255
/sbin/ifconfig eth1 192.168.4.10 netmask 255.255.255.255
/sbin/ifconfig eth2 192.168.1.10 netmask 255.255.255.255
/sbin/route add 192.168.2.20 eth0
/sbin/route add 192.168.4.20 eth1
/sbin/route add 192.168.1.20 eth2
#/sbin/route add default gw 192.168.3.2
```

5.4 Anhang D: Skript zur Namensauflösung

Das folgende Listing zeigt die auf allen Maschinen verwendete Datei `/etc/hosts`.

```
127.0.0.1 localhost
129.187.214.40 pcheger10.nm.informatik.uni-muenchen.de
192.168.9.1 pcheger10.nm.informatik.uni-muenchen.de voip1
192.168.9.130 pcheger10.nm.informatik.uni-muenchen.de voip1
192.168.2.20 mpls2
192.168.3.20 mpls1
192.168.5.20 mpls1
192.168.4.20 voip3
192.168.1.20 voip1

::1 localhost ipv6-localhost ipv6-loopback
fe00::0 ipv6-localnet
ff00::0 ipv6-mcastprefix
ff02::1 ipv6-allnodes
ff02::2 ipv6-allrouters
.ff02::3 ipv6-allhosts
```

Kapitel 6

Literaturverzeichnis

Literaturverzeichnis

[ADF+01] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. RFC 3036: Ldp specification. RFC, IETF, January 2001.

[Atk94] R. Atkinson. RFC 1626: Default ip mtu for use over atm aal5. RFC, IETF, May 1994.

[BBM92] T. Bradley, C. Brown, and A. Malis. RFC 1294: Multiprotocol interconnect over frame relay. RFC, IETF, January 1992.

[DLM+01] B. Davie, J. Lawrence, K. McCloghrie, E. Rosen, G. Swallow, Y. Rekhter, and P. Doolan. RFC 3035: Mpls using ldp and atm vc switching. RFC, IETF, January 2001.

[LH98] M. Laubach and J. Halpern. RFC 2225: Classical ip and arp over atm. RFC, IETF, April 1998.

[NKD+01] K. Nagami, Y. Katsube, N. Demizu, H. Esaki, and P. Doolan. RFC 3038: Vcid notification over atm link for ldp. RFC, IETF, January 2001.

[Plu82] D.C. Plummer. RFC 826: Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. RFC, IETF, November 1982.

[RTF+01] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. RFC 3032: Mpls label stack encoding. RFC, IETF, January 2001.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. RFC 3031: Multiprotocol label switching architecture. RFC, IETF, January 2001.

[TG01] B. Thomas and E. Gray. RFC 3037: Ldp applicability. RFC, IETF, January 2001.

Die im Kriterienkatalog erwähnten Implementierungen wurden auf folgen Seiten gefunden:

- *MPLS-Linux* auf `http://mpls-linux.sourceforge.net/`
- *MPLS* auf `http://www.nortelnetworks.com/corporate/technology/mpls/index.htm`
- *MPLS for Linux* auf `http://cell-relay.indiana.edu/mhonarc/mpls/2000-Jul/msg00196.html`