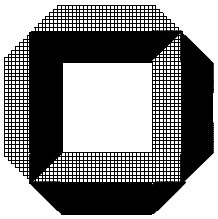


Universität Karlsruhe

Fakultät für Informatik

Institut für Telematik

Transport von NetNews via IP-Multicast

Diplomarbeit von cand. inform. Heiko Rupp

am

Telecooperation Office (TecO)

Institut für Telematik

Universität Karlsruhe (TH)

SS 1997

Betreuer:

Prof. G. Schneider

Dipl.-Inform. Markus Lauff

Ausgabe der Arbeit: 1. April 1997

Abgabe der Arbeit: 30. September 1997

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, September 1997

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Überblick und Einleitung

1.1 Motivation

Die NetNews, eine Art elektronisches Schwarzes Brett, erfreuen sich trotz des Erfolges des World Wide Webs (WWW) weiterhin einer großen Beliebtheit. So werden täglich weltweit mehr als 300.000 so genannte *Newsartikel* in mehr als 10.000 Themengruppen (Newsgruppen) an dieses Brett angehängt.

Da dieses Schwarze Brett nicht auf einem einzelnen Server implementiert ist, sondern als verteilte Anwendung, müssen die Artikel zwischen den beteiligten Rechnern ausgetauscht werden. Dies geschieht im Internet meist in der Form, daß alle Nachbarn, die Interesse an einem speziellen Themengebiet haben, die neuen Artikel angeboten bekommen. Jeder Nachbar entscheidet dann, ob er die neue Nachricht bekommen will oder nicht. Die Zeit, die nötig ist, um einen Artikel von einem Server zum nächsten zu transportieren, liegt dabei in der Größenordnung von wenigen Sekunden.

Diese Art des Austausches der Nachrichten ist schnell und zuverlässig, jedoch wird die zugrundeliegende Kommunikationsinfrastruktur oft nur schlecht ausgenutzt, z.B. indem Nachrichten mehrfach dieselbe physikalischen Leitung überqueren und somit mehr Bandbreite brauchen, als nötig. Dies ließe sich durch den Einsatz zusätzlicher Server am anderen Ende der Leitungen optimieren; dem steht allerdings der finanzielle und personelle Aufwand an zusätzlicher Hardware und Administration entgegen.

Mit IP Multicast [Dee89] existiert eine Lösung, die das Problem bereits in der Netzwerkschicht angeht. Für die Benutzung auf IP Multicastfähigen Netzwerken und speziell auf dem MBone [Dee93, SRL96, Kum96] existieren bereits Protokolle für die Übertragung von Dateien und speziell Multimediadaten, allerdings ist keines dieser Protokolle für die Übertragung von NetNews geeignet.

1.2 Aufgabenstellung

Die Aufgabe ist es, einen alternativen Transportmechanismus für NetNews auf Basis von IP-Multicast zu entwickeln, um so Bandbreite zu sparen. Neben der Protokoll- und Softwareentwicklung steht noch die Beurteilung des Protokolls im praktischen Betrieb.

1.3 Überblick

Diese Sektion gibt einen kurzen Überblick über diese Arbeit. Nach der Motivation und der Aufgabenstellung folgen in Kapitel 2 die nötigen Grundlagen.

Kapitel 3 beschäftigt sich mit dem Konzept, welches hinter dem Transport von NetNews via IP-Multicast steht. Eine Beschreibung der Implementierung folgt in Kapitel 4 und Kapitel 5 zeigt, wie die Software installiert und eingesetzt wird.

Kapitel 6 diskutiert die gewonnenen Ergebnisse. Mit Kapitel 7 und 8 folgen dann noch Ausblick und Danksagungen. Im Anhang befinden sich die Nachdrucke eines Papiers, das zu Jenc8 eingereicht wurde [Rup97], sowie der Protokollspezifikation als Internet Draft.

1.4 Begriffsdefinition *Mcntp*

Im folgenden Text wird oft das Word *Mcntp* auftauchen. *Mcntp* ist die Abkürzung von *M*ulti*c*ast *N*ews *T*ransport *P*rotokoll. *Mcntp* bezeichnet sowohl das Protokoll als auch die entwickelte Software, wobei der spezielle Gebrauch aus dem Kontext hervorgeht.

2 Grundlagen

Damit die Wirkungsweise von *Mcntp* verständlich wird, sollten zunächst ein paar Begriffe eingeführt werden. Rechnerabhängige Begriffe beziehen sich im folgenden immer auf das Unix$^{(TM)}$ Betriebssystem, da zum einen auf diesem Betriebssystem TCP/IP frei verfügbar implementiert wurde und zum anderen Unix eine gute Entwicklungsplattform bildet. Wenn eine Angabe in der Form *Routine(n)* gemacht wird, so bezieht sich diese auf eine Routine, die in Teil *n* der Unix Handbücher gefunden werden kann.

2.1 Die Internetprotokollfamilie

Das Internetprotokoll oder kurz IP ist auf Schicht 3 im OSI-7- Schichtenmodell angesiedelt und dient dazu, ein Datagramm von einem Absenderrechner zu einem Zielrechner zu senden. Ein Datagramm ist hier ein einzelner Datenblock [Pos81a] .

IP abstrahiert völlig vom darunterliegenden Transportmedium, so daß Rechner innerhalb des lokalen Ethernet, via Datex-P oder gar Modem erreicht werden können und somit vom Netzwerkzugang unabhängig sind. Das Internetprotokoll ist in [Pos81a] und im MIL-STD 1777 des amerikanischen Verteidigungsministeriums beschrieben.

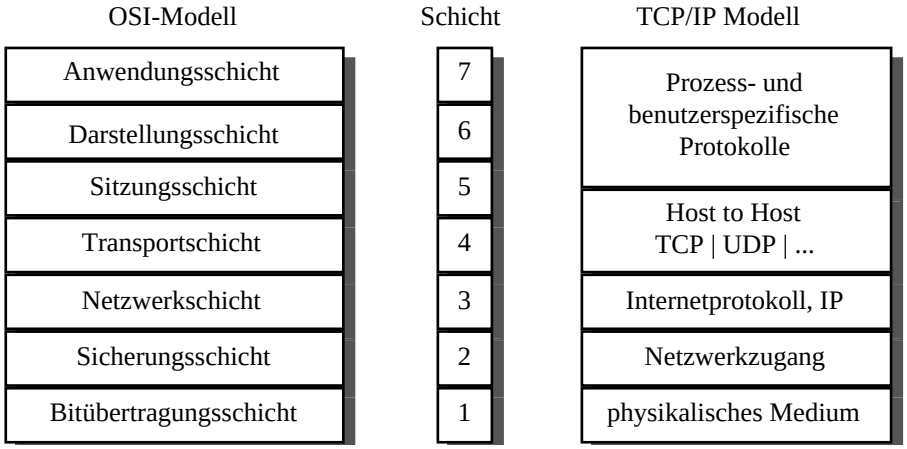


Abbildung 1: Gegenüberstellung ISO und TCP/IP

Auf die Bedeutung der einzelnen Schichten wird hier nicht näher eingegangen, sofern dies nicht für das weitere Verständnis nötig sind. Zu beachten ist , daß in den beiden Modellen eine unterschiedliche Anzahl von Schichten definiert ist; wenn man über die Funktionalität der Schichten redet, muß deshalb auch immer das zugrundeliegende Modell mit angegeben werden. Weitere Informationen gibt es u.a. in [Gib95] oder [P.L92].

2.1.1 IP-Nummern

Jeder Host im Internet muß mindestens eine IP-Adresse haben, die weltweit eindeutig sein muß. In der Version des Internetprotokolls, die zur Zeit in Benutzung ist (IPv4) umfassen IP-Nummern den Bereich von $0 - 2^{32} - 1$ und werden meist im sog. „dotted-quad“ Format als Gruppe von 4 Bytes, die durch Punkte getrennt sind geschrieben (*a.b.c.d*).

Neben den IP-Nummern gibt es außerdem Netzmasken , die angeben, welcher Teil einer IP-Nummer die Adresse eines Netzes und welcher die Hostnummer innerhalb des Netzes ist; die Netzmaske wird auf die gleiche Art und Weise geschrieben, wie die IP-Nummer.

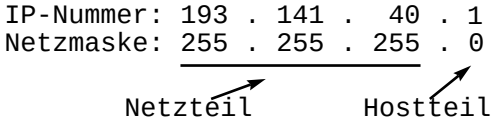


Abbildung 2: IP-Nummer und Netzmaske

Für bestimmte IP-Nummernbereiche wurden Netzmasken vorgegeben und die Netze damit in Klassen eingeteilt, was Routing erleichtert, da man, um ein komplettes Netz zu routen, nur einen Eintrag benötigt und nicht n Stück, die sich nur an je einer Stelle unterscheiden (siehe z.B. `/usr/include/netinet/in.h`).

Die Unterscheidung der Netzklassen erfolgt durch die ersten Bits der IP Nummer, wie im folgenden zu sehen ist. Dabei wird auch implizit eine Netzmaske vergeben.

Netzklasse	linke Bits	Anzahl Netze	Anzahl Hosts pro Netz	Netzmaske
Class A	0	127	ca. 16 Millionen	255.0.0.0
Class B	10	ca. 16000	ca. 65000	255.255.0.0
Class C	110	ca. 4 Millionen	254	255.255.255.0

Tabelle 1: Netzklassen

Neben der Schreibweise einer Adresse als Adresse plus Netzmaske, gibt es seit einiger Zeit noch eine zweite Schreibweise, bei der angegeben wird, wieviele Bits in der Netzmaske von links her gesetzt sind. Diese Schreibweise wird als CIDR (Classless Inter Domain Routing) bezeichnet und kommt aus dem Bereich der IP-Routings. Ein Beispiel: die CIDR-Adresse 193.141.40.0/22 ist äquivalent zur Adresse 193.141.40.0 mit einer Netzmakse von 255.255.252.0 bzw. den vier Class-C Netzen 193.141.40.0 - 193.141.43.0.

In der Praxis hat CIDR inzwischen die implizit vergebenen Netzklassen abgelöst. Dies geschieht, da durch den Internet-Boom die IP-Adressen knapp werden und nicht mehr automatisch ein Class-C Netz vergeben wird, wenn jemand neu an das Internet angebunden werden möchte, sondern nur noch die Anzahl von IP-Adressen, die notwendig sind.

Ausser den oben genannten Klassen gibt es noch die *Class D* Netze, die den IP-Nummernbereich 224.0.0.0 bis 239.255.255.255 umfassen. Die Class D Netze sind als IP-Multicastadressen (siehe 2.2) reserviert. Bei den Class D Adressen kann man nicht, wie bei Class A bis Class C Netzen, von Netzen und Hosts reden, sondern jede Adresse bezeichnet eine einzelne Multicastgruppe.

Im zukünftigen IP-Standard (IPv6) werden die IP-Adressen 16 Byte lang sein. Sie werden als Folge von acht mal zwei Bytes geschrieben, die durch Doppelpunkte getrennt sind (z.B. `FF01:0:0:0:0:0:0:43`), wobei eine Folge von Nullen weggelassen werden kann (das vorige Beispiel könnte also auch als `FF01::43` geschrieben werden); der IPv4-Adressraum wird in den IPv6 Adressraum eingebettet sein (`x:x:x:x:x:x:a.b.c.d` oder verkürzt `::a.b.c.d`), so daß der Umstieg von IPv4 auf IPv6 einfacher vollzogen werden kann [Kus94].

2.1.2 Kommunikationsendpunkte

Um ein Datagramm (s.o.) zu versenden muß für den Datenaustausch beim Absender ein Kommunikationsendpunkt, ein sogenannter Socket eingerichtet werden (siehe socket(2)). Einem Socket sind Eigenschaften wie die Kommunikationsdomäne (z.B. lokaler Rechner oder Internet), der Typ (z.B. Rohdaten oder Benutzerdatagramm) und das verwendete Protokoll zugeordnet. Als Protokolle werden im Internet unter anderem die folgenden verwendet [UCB94a]:

IP ist das Netzwerkprotokoll der Internetprotokollfamilie und befindet sich auf Schicht 3 im OSI-7- Schichtenmodell. Protokolle auf höheren Schichten (wie z.B. TCP und UDP), die auf IP aufsetzen, können Optionen auf IP Ebene setzen. Auf IP kann auch durch einen sog. *raw-socket* zugegriffen werden, um neue Protokolle zu entwickeln oder für Spezialapplikationen. Dieser Zugriff ist nur als Superuser möglich[1]. Siehe auch [Pos81a, Ste94].

ICMP Internet Control Message Protocol – ist das Protokoll, das IP benutzt um Fehler- und Kontrollnachrichten zwischen den Netzwerkinstanzen auszutauschen. ICMP kann durch einen "raw-socket" angesprochen werden, um Netzwerküberwachungs und -diagnosefunktionen auszuführen. Siehe auch [Pos81b, Ste94].

UDP User Datagram Protocol – ein Datagramm-Protokoll, mit dem auch Benutzer Datagramme versenden können. Das Netzwerkdateisystem NFS basiert beispielsweise auf UDP. UDP ist verbindungslos - d.h. der Absender bekommt keine Information, ob das Datagramm angekommen ist — wünscht er eine Bestätigung, so muß der Empfänger diese auf Applikationsebene explizit erzeugen und zurückschicken. In [Gib95] steht hierzu: „Ein Datagramm ist eine Nachricht, die ≪nach bestem Wissen und Gewissen≫ übertragen wird". Siehe auch [Pos80, Ste94]

TCP Transmission Control Protocol – Ein Protokoll, das eine gesicherte und fehlerkorrigierte Duplexverbindung zwischen zwei Endpunkten herstellt. TCP ist verbindungsorientiert: zwischen Sender und Empfänger wird eine Verbindung (Schicht 4) aufgebaut, über die die Daten transportiert werden. Fehlerkorrekturen und Flusskontrolle etc. werden vom Betriebsystemkern erledigt; ebenso wird bei TCP garantiert, daß Pakete in der Reihenfolge ankommen, in der sie abgeschickt wurden. Siehe auch [Pos81c, Ste94].

IGMP Internet Group Management Protocol – Ein Protokoll, welches für das Management von Gruppen von Hosts verwendet wird. IGMP ist für IP Multicast notwendig. Siehe auch [Dee89] und 2.2.2.

Es existieren noch weitere Protokolle, die obigen sind die bekanntesten und für die weiteren Ausführungen die wichtigsten – `/etc/protocols`[2] gibt nähere Auskunft über "übliche" Protokolle.

Nachdem der Socket erfolgreich eingerichtet wurde, muß die Verbindung zur Gegenseite mit Hilfe des connect() Systemaufrufs aufgebaut werden. Dies kann entfallen, wenn nur ein einmaliges Datagramm zu senden ist. Wurde der connect() erfolgreich ausgeführt und besteht die Verbindung, kann auf den Socket mit den normalen read() und write() Systemaufrufen zugegriffen werden.

2.1.3 Dienste im Internet

Die im Internet bzw. von Rechnern im Internet angebotenen Dienste sind beliebig - manche sind fast auf allen Rechnern vorhanden (z.B. smtp - Elektronische Post [Pos82] oder `telnet(1)`), andere nur selten (z.B. archie).

[1] Unless you are extremely clever, socket code works on PPP, Ethernet,IP tunneled into DECnet, FDDI, or carrier pigeons. — Paul A. Vixie
[2] In dieser Datei sind bei UNIX Systemen die bekannten Protokolle aufgeführt.

Was die (meisten) Dienste jedoch gemeinsam haben, ist, daß sie auf definierten Ports zu finden sind. Ports sind protokollabhängige Subadressen innerhalb eines Hosts; sie sind z.B. mit Postfächern zu vergleichen, wobei das Postamt als ein Host mit einer oder mehrerer IP-Adressen angesehen wird. Die bekannten Ports werden, wie viele andere Nummern im Internet von der IANA zentral vergeben, um so die Eindeutigkeit der Nummern zu garantieren – siehe auch Abschnitt 2.6 auf Seite 22.

Der folgende Ausschnitt von `/etc/services` [3] zeigt exemplarisch einige Ports mit den zugeordneten Diensten:

Name	Portnummer	Protokoll	Aliasname
telnet	23	tcp	
smtp	25	tcp	mail
domain	53	tcp	nameserver
domain	53	udp	nameserver
mcntp	5418	tcp	multicast news directory
mcntp	5418	udp	multicast news directory

Tabelle 2: Ein Ausschnitt aus `/etc/services`

Hier sieht man, daß Portnummern durchaus doppelt vergeben werden können, da es sich, wie im Fall des *domain* Dienstes um zwei verschiedene Protokolle handelt. Aus der Tabelle ist zu erkennen, daß auf Port 23 des TCP Protokolls ein *Telnet* Dienst zu erwarten ist, sofern dieser Port bedient wird. Diese Ports sind also die Kommunikationsendpunkte auf der anderen Seite einer Kommunikationsverbindung. Ein Serverprozeß hat auch einen Socket geöffnet und wartet nun mit diesem Socket, bis eine Verbindung zu ihm aufgebaut wird.

Ein bestimmter Dienst ist durch das folgende Tripel repräsentiert:

Dienst = (Host,Port,Protokoll)

Dieses Tripel ist eindeutig und es können keine zwei unterschiedliche Dienste mit dem gleichen Tripel existieren. Den symbolischen Namen für einen Dienst kann man im Normalfall in `/etc/services` finden.

Prinzipiell werden Ports auf Unix$^{(TM)}$ Rechnern für ausgehende Verbindungen – für das Ziel muß der Zielport angegeben werden – dynamisch, mit niederen Nummern beginnend, vergeben: ein Prozeß, der einen Port zur Kommunikation benötigt, bekommt den nächsten freien mit der niedrigsten verfügbaren Nummer vom System zugewiesen. Damit aber kein Benutzer dies als Sicherheitsloch ausnutzen kann, indem er sich sehr früh einloggt, oder falls ein alter Port nicht mehr benötigt wird, diesen wiederbenutzt und einen eigenen Server auf diesen Port setzt, dürfen Ports mit einer Nummer ≤ 1024 nur von `root` belegt werden; `root` ist per Definition der Benutzer auf einem Unix System, der vertrauenswürdig ist[4]. Wäre diese Einschränkung nicht gegeben, könnte nicht mit Sicherheit gesagt werden, welcher Dienst auf welchem Port zu erwarten ist. Aus diesem Grund sind die Dienste fest den Ports zugeordnet und wenn ein Server auf Port *n* arbeitet, kann zu diesem Port eine Verbindung aufgebaut werden. Manche Server (wie z.B. X Windows oder Irc) nehmen sich Ports mit einer hohen Nummer (6000 und 6667), so daß dort die Wahrscheinlichkeit der Portblockierung zum Zeitpunkt des Serverstarts gering ist.

[3] In dieser Datei sind bei UNIX die bekannten Ports aufgeführt.

[4] Diese Annahme stammt aus einer Zeit, als Unix Rechner teuer waren und von speziell geschultem Personal administriert wurde. Heutzutage gibt es für nahezu jede Einbenutzermaschine einen TCP/IP Protokollstapel, so daß die hier Kontrolle durch den Administrator nicht mehr erfolgen kann.

2.2 IP-Multicast

Neben den Verbindungstypen Punkt-zu-Punkt (Unicast) und Einer-an-alle (Broadcast) gibt es noch den Typ Einer-an-mehrere: *Multicast*.

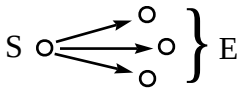


Abbildung 3: Multicast: Ein Sender mehrere Empfänger

Steeve Deering beschreibt in [Dee89], welche Erweiterungen an der IP-Software nötig sind, um IP-Multicast zu unterstützen. Mit der Hilfe von Routingprotokollen (siehe [WPD88, Moy94]) und spezieller Routingsoftware werden Pakete bei Multicast nur an die Empfänger geschickt, die diese Informationen auch bekommen möchten. Hierzu muß ein System eine sog. Multicastgruppe abonnieren. Die Router geben diese Information dann weiter, bis ein Weg vom Sender zum Empfänger besteht. Im Gegensatz zu TCP-Verbindungen [Pos81c], beschränkt sich IP-Multicast auf Datagramme, so daß die Übertragung verbindungslos ist und “mit besten Kräften” durchgeführt wird. Dabei kann es passieren, daß unbemerkt Pakete verloren gehen, doppelt am Ziel ankommen oder in der Reihenfolge vertauscht werden [Pos80].

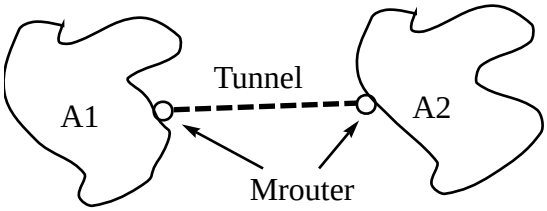


Abbildung 4: Multicastfähige Inseln werden verbunden

Manche ältere Betriebssysteme und Router haben noch keine Unterstützung für IP-Multicast eingebaut. Sollen nun zwei Gebiete, *A1* und *A2* (siehe Abbildung 4), zu einem großen Gebiet zusammengefaßt werden, müssen die Multicastpakete mit speziellen Routern, *Mroutern*, zum anderen Gebiet transportiert werden. Dabei werden die Pakete in normale IP-Pakete eingepackt und am Ziel-Mrouter wieder ausgepackt; diesen Vorgang nennt man *tunneln* und die Verbindung zwischen den beiden Mroutern *Tunnel*.

2.2.1 Zuverlässiges Multicast

Um der Hauptschwäche des normalen Multicast, der Tatsache, daß die Verbindungen ungesichert sind, zu entgegnen, wurden Protokolle entwickelt, die gesicherte Verbindungen zur Verfügung stellen, ähnlich wie TCP dies auch tut.

Der naheliegende Ansatz ist, Pakete, die beim Empfänger angekommen sind, vom Empfänger bestätigen zu lassen. Kommt die Bestätigung vom Empfänger nicht innerhalb einer gewissen Zeitspanne zurück, so schickt der Sender das Paket einfach noch einmal. Dies kann aber sehr schnell zu Problemen führen, da beim Multicastsender zu

viele Bestätigungspakete ankommen und dadurch dessen Leistungsfähigkeit sinkt.

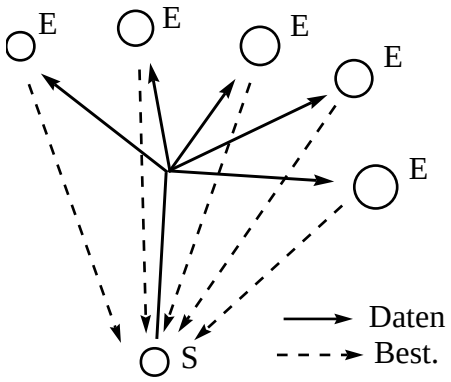


Abbildung 5: Implosion

Im Beispiel in Abbildung 5 kommen für jedes ausgesandte Multicastpaket fünf Antwortpakete zurück, die verarbeitet werden müssen; dieser Vorgang wird als *Implosion* bezeichnet. Ab einer gewissen Anzahl von Multicastempfängern übersteigt das Volumen der Bestätigungen das der verschickten Daten.

RMTP

Bei RMTP [LS95] wird der Verringerung der Leistungsfähigkeit vorgebeugt, indem sogenannte “Designated Receivers” (*dr*) in die Hierarchie eingefügt werden. Die *dr* empfangen die Daten ebenfalls vom Sender und speichern diese lokal zwischen. Die Empfänger senden ihre Bestätigunspakete an die *dr*. Die *dr* sammeln diese und senden, wenn sie alle Bestätigungen ihrer lokalen Gruppenmitglieder erhalten haben, eine Bestätigung an den Sender. Wenn ein Mitglied der lokalen Gruppe ein Paket nicht erhalten hat, schickt es ihm der *dr* nach; hat dieser das Paket auch nicht erhalten, fordert er es beim Sender erneut an.

In der untenstehenden Grafik erhält der Sender nur noch zwei Bestätigungspakete pro versandtem Datenpaket.

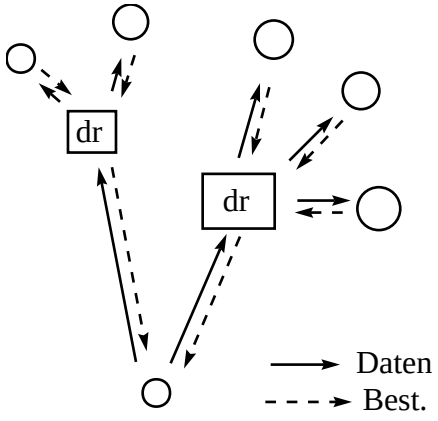


Abbildung 6: RMTP

Großer Nachteil ist, daß für die *dr* spezielle Hard- und Software bereitgestellt werden muß. Diese muß über genug Speicher verfügen, so daß die Datenpakete so lange zwischengespeichert werden können, bis alle Bestätigungen

erhalten wurden. Außerdem ist die Zuverlässigkeit nur garantiert, solange alle *dr* funktionieren. Fällt ein *dr* aus, kann zwar rekonfiguriert werden, aber in der Zwischenzeit ist die Übertragung unsicher.

Der Beschreibung von RMTP ([LS95]) nach scheint RMTP sich am besten für einmalige Operationen zu eignen, wie z.B. eine Implementierung von rdist(1) via Multicast, da die Daten zwischengespeichert werden, bis die letzte Bestätigung erhalten wurde.

XTP

Mit XTP existiert ein Protokoll, das sowohl auf IP, als auch auf ATM aufgesetzt werden kann und das unter anderem auch ein “gesichertes” Multicasting zulässt; d.h. die Empfänger können Datenpaktete erneut anfordern, wenn sie feststellen, daß ihnen eines fehlt. XTP wäre eine mögliche Alternative zu normalem IP-Multicast; allerdings muß dafür XTP für alle unterstützten Plattformen frei verfügbar sein. Dies scheint gegeben, da XTP ein offener Standard ist [SDW92].

Wenn gesichertes Multicast gewünscht wird, muß der Sender Fehlerpakete erneut nachsenden, was zu der oben erläuterten Implosion führen kann. Werden wie bei KLG [Hof94] lokale Gruppen gebildet, die ähnlich der *dr* bei RMTP eine “lokale Fehlerbehandlung” durchführen, muß widerum spezielle Software zum Einsatz kommen. Außerdem müsste die Kommunikation der Gruppenmitglieder in einer lokalen Gruppe über WAN-Leitungen gehen, so daß diese wieder stärker belastet würden, was aber vermieden werden soll.

Ein großer Vorteil von XTP gegenüber IP ist, daß das Protokoll auf höhere Transportgeschwindigkeiten optimiert wurde und deswegen bei schnellen Netzwerktechnologien, wie ATM oder FDDI besser einsetzbar ist; dieser Vorteil spielt aber nur dann eine Rolle, wenn die Sicherung angeschaltet ist, da das Fenster innerhalb dessen eine Bestätigung vorliegen muß, größer ist als bei TCP. Wenn nur ungesicherte Pakete versendet werden, ist XTP gleichwertig zu UDP. Weiterhin muß XTP, um überhaupt Multicast unterstützen zu können, auf normalem IP-Multicast aufsetzen.

2.2.2 Multicast Sockets

Ein Kommunikationsendpunkt im TCP/IP Protokollstack wird Socket genannt. Die für IP Multicast verwendeten Sockets unterscheiden sich von den Unicast- oder Broadcastsockets darin, daß zusätzliche Funktionalität für das Betreten und Verlassen von Gruppen (join und leave) vorhanden ist. Außerdem werden Pakete, wenn mehrere Prozesse auf einem Host Mitglied in einer Multicastgruppe sind, an alle Prozesse ausgeliefert. Besitzt ein Host Sender- und Empfängerprozesse, so erhalten diese Empfängerprozesse die vom Sender verschickten Daten ebenfalls; dieses Verhalten kann aber auf Wunsch geändert werden.

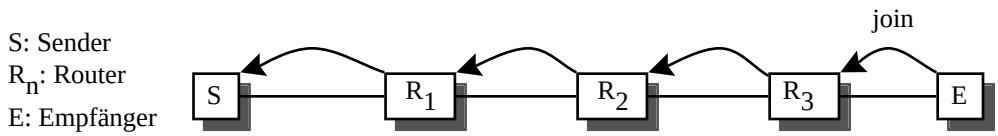


Abbildung 7: Join Nachrichten wandern bis zum Sender

Wenn ein Prozeß Mitglied in einer Multicastgruppe wird, sendet das IP-Subsystem eine IGMP [Dee89] Nachricht an die Multicastgruppe 224.0.0.1 (ALL-HOSTS.MCAST.NET), daß nun diese neue Gruppe benötigt wird. Alle multicastfähigen Router in diesem Netzsegment geben diese Mitgliedschaftsanfrage nun an ihre benachbarten Router weiter und tragen den Weg zum Empfänger in ihre Tabellen ein, bis ein Weg vom Sender zum Ziel existiert.

Wird die Multicastgruppe wieder verlassen, bauen die Router diesen Weg wieder ab, vorausgesetzt, daß dieser Host der letzte war, der die Gruppe verlassen hat.

Da das Installieren und Abbauen der Routen eine gewisse Zeit benötigt, wird ein Empfänger, der eine Gruppe betritt, nicht immer sofort Pakete vom Sender erhalten, sondern erst nach einer kurzen Verzögerung; Pakete die in der Zwischenzeit vom Sender verschickt wurden, können also den Empfänger nicht unbedingt erreichen. Dies hängt davon ab, inwieweit die Router auf dem Weg Pakete zwischenpuffern können, wie nahe der Empfänger an einem Router ist, bei dem diese Gruppe schon bekannt ist und nicht zuletzt davon, mit welcher Rate der Sender Pakete verschickt.

Dadurch, daß die Verteilungsbäume dynamisch auf- und abgebaut werden, erhalten nur die Zweige Daten, die diese auch wirklich erhalten wollen.

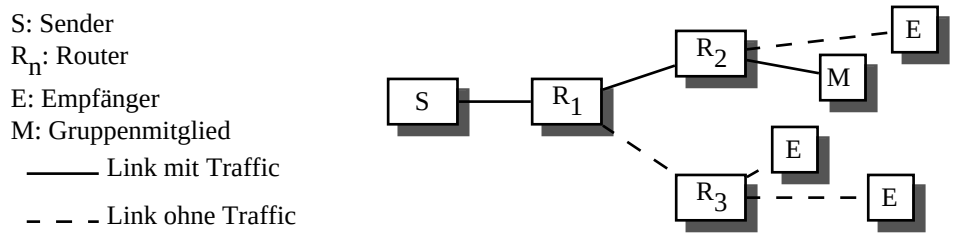


Abbildung 8: Nur Links, die Daten wollen, bekommen diese

In der Abbildung 8 erhalten nur die Router Multicastdaten vom Sender S die auf dem Weg zum Gruppenmitglied M liegen.

2.2.3 Multicast und TTL

Wie alle IP Pakete erhalten Multicastpakete ebenfalls eine TTL (Time to Live), die angibt, wie weit sich ein Paket ausbreiten darf. Die TTL wird in jedem Router, durch den ein Paket fließt um eins dekrementiert. Ist die TTL auf Null gesunken, wird das Paket verworfen. Dies geschieht u.a., um Pakete, die auf Grund von Routingfehlern immer zwischen zwei Rechnern hin- und herwandern aus dem Netz entfernen zu können. Im Gegensatz zu Unicastpaketen werden Multicastpakete nicht nur dann verworfen, wenn die TTL auf Null gesunken ist, sondern auch dann nicht weitergeleitet, wenn die TTL niedriger als ein gewisser, am Router einzustellender, Schwellenwert ist. So können Multicastgruppen lokal zu einer Organisation oder eines definierten Bereiches gehalten werden.

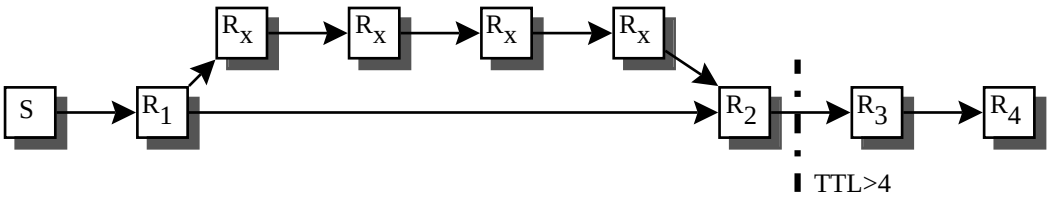


Abbildung 9: Beschränkung der Ausbreitung von Multicastpaketen mittels TTL

Bei der in Abbildung 9 gezeigten Anordnung wird Router R_2 Pakete nur dann an Router R_3 weiterleiten, wenn die TTL der Pakete größer als vier ist.

Die nächste Tabelle zeigt, wie eine mögliche Zuordnung von TTL Werten zu Reichweiten definiert werden könnte (aus [UCB94b, Seite 34]):

Reichweite	Start TTL
Gleicher Host	0
Gleiches Subnetz	1
Gleiche Ort	32
Gleiche Region	64
Gleicher Kontinent	128
Unbeschränkt	255

Tabelle 3: Mögliche Reichweiten von Multicastpaketen

Hierbei sind die Bezeichnungen “Ort” und “Region” nicht strikt definiert und können bei Bedarf auch weiter unterteilt werden.

Ein Problem der Ausbreitungsbeschränkung via TTL, welches besonders im Zusammenhang mit den Multicast-routingprotokoll DVMRP[5][Pus96] auftritt ist folgendes (diese Ausführug bezieht sich auf Abbildung 9): Angenommen S sendet ein Paket mit einer Start-TTL von zehn. Dieses Paket läuft den “oberen” Weg über R_1, vier mal R_x und R_2 nach R_3. Bei R_3 hat es eine Rest-TTL von drei und wird verworfen. Nun kann es aber sein, daß ein Paket, das von S ebenfalls mit einer Start-TTL von zehn verschickt wurde den “unteren” Weg über R_1 und R_2 nach R_3 nimmt. Dieses Paket hat bei R_3 noch eine TTL von sieben und wird in Richtung R_4 weitergeleitet. Damit darf also R_3 den Multicastroutingbaum für diese Gruppe nicht beschneiden, selbst wenn Pakete wegen zu niedriger Rest-TTL nicht weitergeleitet werden, denn es könnte sein, daß Pakete ankommen, die eine Rest-TTL haben, die groß genug ist und die deswegen weitergeleitet werden müssen.

2.2.4 Multicastadressbereiche

Wie oben (Abschnitt 2.1.1 auf Seite 3) beschrieben, ist der Adressbereich von 224.0.0.0 - 239.255.255.255 für Multicastadressen reserviert. Jede Multicastadresse bezeichnet dabei eine Gruppe von Hosts, die Pakete erhalten, die an diese Gruppe gesandt werden. Der Adressraum ist in weltweit und lokal gültige Gruppen unterteilt (siehe auch [Mey97]):

- Global gültige Adressen. Diese Adressen sind weltweit gültig und können von jedem benutzt werden. Dieser Adressbereich ist weiter aufgeteilt:
  - Adressen, die von der IANA vergeben werden (224.0.1.0/24).
  - Adressen, die jeder verwenden kann (225/8 - 238/8).
- Lokale Adressen. Auch hier gibt es nochmals eine Unterteilung:
  - Adressen, die lokal für ein Subnetz sind (224.0.0.0/24). Dies schließt Adressen, wie z.B. “Alle Hosts im Subnetz” (ALL-SYSTEMS.MCAST.NET) oder “Alle Router im Subnetz” (ALL-ROUTERS.-MCAST.NET) ein. Diese Adressen werden von der IANA (siehe Abschnitt 2.6 auf Seite 22) vergeben.

[5] DVMRP ist das Routingprotokoll, welches aktuell im MBone verwendet wird

– Adressen, die lokal für ein Standort (239.255.0.0/16) oder eine Organisation (239.192.0.0/14) sind. Für diesen Adressbereich gelten Regeln, die in [Mey97] definiert sind:
  * Bereiche mit gleicher Gültigkeit dürfen sich nicht überlappen
  * Der Bereich eines Standortes darf nicht größer sein, als der einer Organisation
  * Die Router an den Rändern eines Bereiches dürfen keine Pakete von außen in diesen Bereich und von innen aus dem Bereich heraus weiterleiten.

  Diese Adressbereiche werden auch als *administrativ beschränkte Bereiche* bezeichnet.

Bei den Multicastadressen, die jeder verwenden kann, ist die Möglichkeit gegeben, daß zwei Sender A und B, die geographisch und netztopologisch weit auseinander liegen, dieselbe Multicastgruppe für lokale Empfänger nutzen wollen. Durch die Multicastroutingprotokolle wird nun eine Verbindung zwischen diesen beiden Bereichen geschaffen, so daß Daten von A zu B und umgekehrt fliessen. Dadurch wird Verkehr geschaffen, der unnötig ist:

- Die Pakete fliessen so lange, bis die TTL auf Null gesunken ist. Dies kann weit über den jeweiligen Bereich hinaus sein.
- Wenn die Empfänger im jeweils anderen Bereich nicht auf dem passenden Port warten, wirft das Empfängersystem die empfangenen Daten einfach fort. Allerdings kann es die Multicastgruppe auch nicht abbestellen, weil ein Empfänger für diese Gruppe vorhanden ist.

Um Gruppen, die nur eine limitierte Ausbreitung haben sollen, zu verteilen eignen sich deshalb die Adressen aus dem administrativ beschränkten Bereich, da die beteiligten Router die Pakete nicht aus diesem Bereich hinaustransportieren dürfen.

Eine Liste der Adressen, die von der IANA vergebenen Multicastadressen kann man unter ftp://ftp.isi.edu/in-notes/iana/assignments/multicast-addresses finden.

2.3 Netnews

Die NetNews sind Diskussionsforen, die als eine Art weltweit verteilte schwarze Bretter fungieren. Wenn ein sogenannter Artikel an eines dieser Bretter geheftet wird, erscheint er auch (mit Ausnahmen, siehe unten) auf den anderen Brettern. Diese weltweite Menge von Systemen, die miteinander NetNews austauschen, wird auch als *Usenet* bezeichnet.

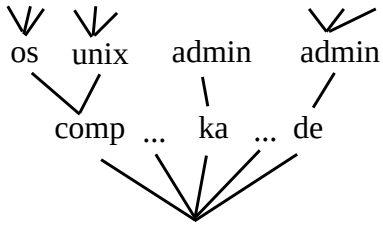


Abbildung 10: Hierarchische Anordnung der Newsgruppen

Die schwarzen Bretter sind nach Themen unterteilt; die einzelnen Unterteilungen werden als Newsgruppen bezeichnet. Der Gruppenname gibt dabei das Thema an, über das in dieser Gruppe[6] diskutiert werden soll (Die Gruppe de.rec.motorrad beschäftigt sich beispielsweise mit allen Aspekten des Motorradfahrens).

[6] es gibt für fast jedes erdenkliche Thema eine Newsgruppe. Fehlt eine, so gibt es immer jemanden, der sie kreiert – hierfür gibt es im Netz feste Prozeduren, die es zu befolgen gilt, ansonsten wird die Gruppe von den Administratoren der Server nicht beachtet

Die Newsgruppen sind hierarchisch untergliedert – so gibt es neben den klassischen Hierarchien, wie z.B. *comp.** (Themen, die mit Computern zu tun haben) und *sci.**[7] (Diskussionen über diverse wissenschaftliche Themen) auch lokale und nationale Hierarchien (*ka.** oder *de.**). Diese Hierarchisierung dient einerseits dazu, Gruppen mit verwandten Themen zusammenzufassen, andererseits aber auch um die Verteilung zu beschränken (*ka.** ist lokal für Karlsruhe) oder eine sprachliche Abgrenzung der Inhalte herbeizuführen (*fr.** ist französisch-, *de.** deutschsprachig). Die Limitierung der Ausbreitung nach Hierarchienamen funktioniert jedoch in der Praxis nur sehr schlecht. Siehe auch [OT92] und [Har95].

Die einzelnen Artikel müssen in einem festen Format folgen, das in RFC[8] 1036[9] [HA87] festgelegt ist (RFC 1036 bezieht sich selbst dabei auf RFC 822). Hier ein Beispiel:

```
Path: pilhuhn.de!snert!hwr
From: hwr@pilhuhn.de (Heiko W.Rupp)
Newsgroups: ka.uni.rz
Subject: Re: Sauereinen auf der UNI-Homepage??
Date: 5 Jun 1997 08:23:44 GMT
Message-ID: <5n5t2g$4v0$1@snert.pilhuhn.de>
References: <5mc82h$4lj$1@nz12.rz.uni-karlsruhe.de>
Organization: The Home of the Pilhuhn

In <5n4q7h$hue$1@nz12.rz.uni-karlsruhe.de>, Frederik Ramm wrote:
:|berichtet hat, war es wohl wirklich ein Cache-Problem. Also duerfte
:|das Xlink-Usageprofil auch nicht anders aussehen als unseres...

Wieso auch ..
[...]
```

Abbildung 11: Ein Newsartikel

Ein Artikel besteht aus zwei Teilen, die durch eine Leerzeile getrennt sind: Dem Kopf (Header) und den Daten (Body). Die Felder im Header sind in der folgenden Form:

Feld ::= Feldname ":" Leerzeichen Wert

Einige Felder im Kopf müssen zwingend vorhanden sein (From, Newsgroups, Path, Message-ID, Date und Subject) während andere fakultativ sind. Bis auf die Newsgroups- und Path- Einträge und die Message-ID werden die Felder hier nicht weiter erläutert, sondern auf RFC 1036 [HA87] verwiesen.

- Im *Newsgroups:* Feld stehen alle Newsgruppen, in denen der Artikel erscheinen soll. Im Normalfall ist dies genau eine Gruppe, es können aber auch mehrere angegeben sein – in diesem Fall spricht man von *Crossposting*. Ein Crossposting wird auf einem Server nur einmal im Dateisystem abgelegt; für den Benutzer scheint es aber so, als ob er in allen Gruppen vorhanden wäre. Crosspostings sparen also gegenüber mehreren Einzelpostings desselben Artikels in verschiedene Gruppen Bandbreite und Plattenplatz.

[7] Die Schreibweise mit dem "*" in comp.* soll andeuten, daß es sich um eine Hierarchie handelt, im Gegensatz zu Einzelgruppen, wie z.B. de.rec.motorrad

[8] Request for Comment. Siehe Abschnitt 2.5 auf Seite22

[9] Momentan wird in der IETF (Internet Engineering Task Force) an einem Nachfolgedokument zu RFC 1036 gearbeitet

- Die *Message-ID* eines Artikels dient der Identifikation des Artikels und muß deshalb weltweit eindeutig sein. RFC 1036 schlägt die folgende Form vor:

  Message-ID ::= “<” lokal-eindeutig “@” Domain “>”

  hierbei dürfen weder *lokal-eindeutig*, noch *Domain* die Zeichen “<”, “>”, Leerzeichen oder “@” enthalten. Beispiele für gültige Message-IDs finden sich in der obigen Abbildung (Abb 11) in den *Message-ID* und *References* Zeilen.

- Der *Path:* eines Artikels gibt an, welche Systeme er passiert hat, um zum aktuellen System zu gelangen. Dabei sind die einzelnen Rechner durch Satzzeichen (außer dem Punkt) getrennt; in der täglichen Praxis hat sich das Ausrufungszeichen bewährt. Die letzte Komponente (ganz rechts) ist dabei der Benutzer, der den Artikel geschrieben hat. Ganz links steht das System, das der Artikel als letztes passiert hat. Jedes System fügt vor der Weiterverteilung seine Kennung dem Path: Header hinzu. Die Hauptbedeutung hat der *Path:* Eintrag in der Optimierung der Newsverteilung (siehe nächster Abschnitt).

Neben den normalen Artikeln gibt es noch solche, die einen *Control:* Header haben. Diese Kontrollnachrichten dienen dazu, den Newssystemen, die den Artikel erhalten, mitzuteilen, eine bestimmte Aktion auszuführen, die als Wert im Headers mitgegeben wird. Aus der Liste der möglichen Kontrollnachrichten[10] sei hier stellvertretend die *Cancel* Nachricht vorgestellt. Mit der Cancel-Nachricht wird dem Newssystem mitgeteilt, den Artikel, dessen Message-ID angegeben ist, zu löschen und nicht mehr weiterzuverbreiten. Eine Überprüfung, ob der Sender der Cancel-Nachricht dazu auch berechtigt ist, findet nicht statt. Andere Kontrollnachrichten fordern das Newssystem dazu auf, die aktuelle Version oder Konfiguration an den Sender der Kontrollnachricht via EMail zu verschicken.

2.3.1 Größenverteilung der Artikel

Um die Größenverteilung von NetNewsartikeln beurteilen zu können, wurden im November 1996 auf verschiedenen Newsservern (Ausnahme Host Snert; hier wurde von November 1996 bis Juni 1997 gemessen) die Größe aller Artikel, die durch das System flossen, mitprotokolliert. Nach dem Aufsummieren ergibt sich die Verteilung in Tabelle Nr. 4 auf Seite 15 (sind Einträge leer, so wurde für diese Kategorie auf dem speziellen Host keine Messung vorgenommen).

Die Rechner, auf denen die Größenverteilung ermittelt wurde sind im einzelnen:

Xlink Blackbush.xlink.net

Switch Swsbe6.switch.ch

Uni-Karlsruhe News.rz.uni-karlsruhe.de

Snert Snert.pilhuhn.de

Wie man leicht sieht, sind mit Ausnahme der Gruppen, in der viele Binärdateien gepostet werden[11], 90% aller Artikel kleiner als 64kBytes. Damit passen 90% aller Artikel in ein einziges UDP Paket. Durch Datenkompression lässt sich diese Anzahl noch erhöhen.

Tabelle 4 auf Seite 15 zeigt, wie die Größen der Artikel verteilt sind, aber leider nicht, welchen Anteil sie am Gesamtvolumen haben. Tabelle 5 auf Seite 16 die auf den Hosts *Snert = snert.pilhuhn.de* und *Blackbush = blackbush.xlink.net* ermittelt wurden, gibt darüber stellvertretend Auskunft.

[10] Siehe RFC 1036 Abschnitt 3
[11] alt.binaries.*

		Angaben in %					
Gruppen	Größe	Xlink 1.Lauf	Xlink 2.Lauf	Xlink 3.Lauf	Switch	Uni-Karlsruhe	Sner t
all.all	<5001	93.13	94.31	93.32	93.39	93.13	
	<9001	96.60	96.90	95.92	95.72	95,62	
	<64001	99.22	99.31	98.85	98.57	98.75	
Ø in Bytes		3397	3150	3984	6074	5068	
alt.all	<5001	86.77	88.41	87.23			
	<9001	91.99	91.85	90.60			
	<64001	97.68	97.40	96.64			
Ø in Bytes		6662	6551	7797			
alt.all,!alt.bin*	<5001	89.65	91.40	91.60			
	<9001	95.30	95.01	95.09			
	<64001	99.01	99.01	98.92			
Ø in Bytes		4024	3900	4003			
alt.bin*	<5001	60.64	58.34	58.53			
	<9001	61.97	59.83	60.99			
	<64001	82.59	80.70	81.35			
Ø in Bytes		30750	33500	32921			
comp.*	<5001	95.47	97.80	97.29			
	<9001	99.19	99.21	98.73			
	<64001	99.93	99.92	99.93			
Ø in Bytes		2056	1892	2058			
de.*	<5001	85.50	95.24	93.63			
	<9001	97.73	97.13	96.03			
	<64001	99.28	99.07	98.38			
Ø in Bytes		3770	3394	4456			
Gemischt,	<5001						95.07
quasi	<9001						97.36
Endsystem	<64001						99.61
Ø in Bytes							2659

Tabelle 4: Größenverteilung von Newsartikeln

Bereich (Bytes)	# Artikel	% Artikel	$\sum$ % Artikel	Volumen	% Volumen	$\sum$ %Volumen
Snert						
<5001	1293724	95.07	95.07	1961 MB	56.85	56.85
5000<9001	31185	2.29	97.36	193 MB	5.60	62.45
9000<640001	30656	2.25	99.61	718 MB	20.82	83.27
>64000	5313	0.39	100.00	577 MB	16.73	100.00
Total	1360878	100.00		3449 MB	100.00	
Blackbush						
<5001	705584	94.48	94.48	1020 MB	41.27	41.28
5000<9001	16386	2.19	96.67	100 MB	4.06	45.33
9000<640001	17578	2.35	99.02	449 MB	18.16	63.50
>64000	7314	0.98	100.00	902 MB	36.50	100.00
Total	746862	100.00		2473 MB	100.00	

Tabelle 5: Volumenverteilungen von Newsartikeln auf den Hosit *Snert* und *Blackbush*

Interessant in diesem Zusammenhang ist noch, daß der Artikelbereich, in dem die meisten Artikel auftauchen in der Größenordnung von 600 - 1200 Bytes ist.

2.3.2 Transport im Internet

Der Transport der Newsartikel erfolgt nach einem optimierten Flächenfüllalgorithmus: Alle Nachbarn, die an einem Artikel Interesse haben könnten, bekommen den Artikel angeboten, außer ihr Eintrag erscheint schon im *Path:* Header. Im obigen Beispiel (Abbildung11) würden die Hosts *snert* und *pilhuhn.de* von einem anderen Newsserver diesen Artikel nicht mehr angeboten bekommen. Die Entscheidung, ob ein Nachbar Interesse an dem Artikel haben könnte, erfolgt durch Vergleich des Newsnamens des Nachbarn mit den Einträgen im Path: Header.

Früher wurden die Newsartikel zwischen den am Usenet beteiligten Rechner via UUCP(1)[12] übertragen. Hierbei wurden Referenzen zu den Artikeln in einer Datei pro Nachbar gespeichert. Regelmäßig (meist stündlich) wurden dann die Artikel gesammelt und zu großen Paketen zusammengefügt, die dann übertragen wurden. Auf Wunsch wurden die Pakete vor der Übertragung noch komprimiert. Dieses Verfahren ist zwar effektiv, hat aber auch den Nachteil, daß selbst Artikel übertragen werden, die auf dem Zielsystem schon vorhanden sind.

Inzwischen erfolgt der Transport der Artikel zwischen den größeren Rechnern (fast) nur noch via *NNTP* (Net News Transfer Protocol, [KL86]). Um einen Artikel via NNTP zu übertragen, baut der sendende Rechner eine TCP Verbindung zum Empfänger (normalerweise Empfängerport 119) auf und fragt die Gegenseite, ob sie an dem Artikel Interesse hat. Falls dies zutrifft, wird der Artikel übertragen.

Möchte beispielsweise Host *S* an Host *E* einen Artikel senden, können NNTP Dialoge, wie in Abbildung 12 auf Seite 17 gezeigt ablaufen.

[12] Unix to Unix Copy Program

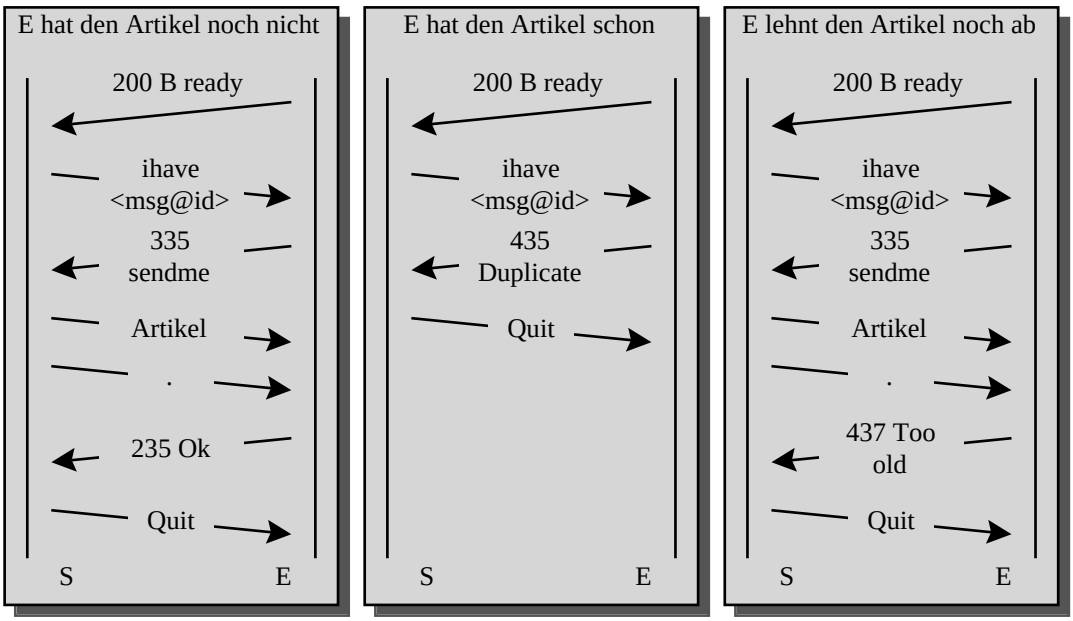


Abbildung 12: NNTP Dialoge

E meldet sich als erstes mit einer Begrüßungsmeldung. S schickt dann das Kommando *ihave*, dem als Parameter die Message-ID mitgegeben wurde. Ist der Artikel auf System E schon vorhanden, antwortet dies, daß es den Artikel schon hat (Code 435) und der Artikel wird nicht übertragen. Ansonsten wird der Artikel übertragen und als Endekennung ein einzelner Punkt auf einer Zeile gesendet. Mit dem Kommando *quit* wird dann die Verbindung wieder abgebaut. Eine Liste ausgewählter NNTP Antwortcodes zeigt die nächste Tabelle:

200	Begrüßungsnachricht
235	Artikel wurde erfolgreich übertragen
335	Ok, schicke den Artikel
400	Server kann keine Artikel annehmen
435	Server hat den Artikel schon

Tabelle 6: NNTP Antwortcodes

Weitere Fehlercodes und die genauen Definitionen können RFC 977 [KL86] entnommen werden.

2.3.3 Artikellaufzeiten

Mit nntplink(8) und innfeed(8) (siehe auch 2.3.4) lassen sich Transferzeiten von deutlich unter einer Sekunde zwischen zwei Systemen erzielen. Aus dem README zum NTA[13] INN:

> „Remember to tell your kids about the days when USENET was store and forward.“ – Jim Thompson, as part of a message that said he was getting under 200ms propagation, disk to disk.

[13] News Transfer Agent — ein News Transport System. Diese Bezeichnung ist analog zu der des MTA (Message Transfer Agent).

Bei angenommen 0,5 Sekunden pro *Hop* (der Transport von einem System zum nächsten) ergeben sich die Werte in Tabelle 7:[14]

	Anzahl Artikel	ØHops / Artikel	ØLaufzeit
Snert	101.918	9.72	5 Sek

Tabelle 7: Zurückgelegte Wege von Newsartikeln

Diese Transferzeiten lassen sich in der Praxis nicht immer realisieren, da hier die Größe des Artikels nicht berücksichtigt wurde und auch die Last auf den Servern außer Acht gelassen wurde. Oft werden Server von den Zugriffen auf die Plattenspeicher ausgebremst. Außerdem spielt auch noch die Netzwerkanbindung der beiden Partner eine Rolle. Um einen Artikel via NNTP zu übertragen, müßen Datenpakete zweimal den Weg vom Sender zum Empfänger und zurück wandern.

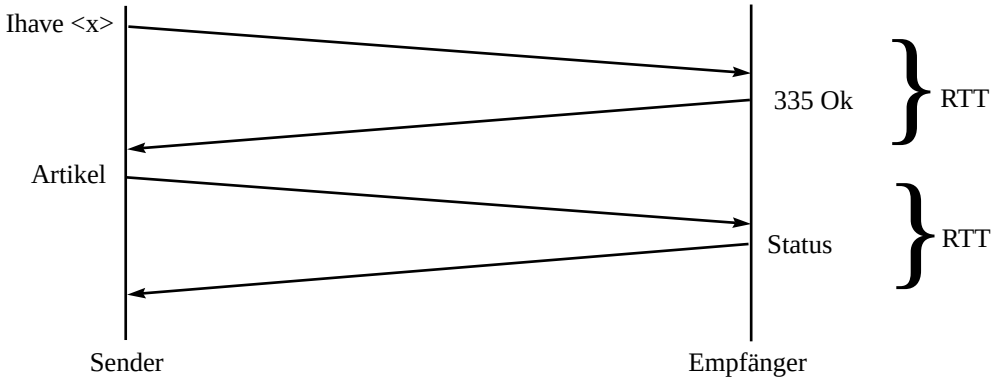


Abbildung 13: Paketrundlauzeit

Die Zeit für eine Runde wird auch als Paketrundlauzeit oder Round Trip Time (RTT) bezeichnet und kann mit Hilfe von ping(8) wie folgt ermittelt werden:

```
snert# ping -c 2 blackbush
PING blackbush.xlink.net (193.141.40.8): 56 data bytes
64 bytes from 193.141.40.8: icmp_seq=0 ttl=253 time=160.074 ms
64 bytes from 193.141.40.8: icmp_seq=1 ttl=253 time=160.312 ms
```

Abbildung 14: Messung der Paketrundlaufzeit

Ein Paket vom Rechner *snert* zum Rechner *Blackbush* benötigt also in diesem Fall ca. 160ms und somit der NNTP Transfer eines Artikels mindestens 320ms. Hierzu kommen noch die Zeit für die Bearbeitung der ihave-Anfrage und des Artikels selbst, um den Status des Transfers an den Sender zurückmelden zu können.

Streaming-NNTP

Eine Verbesserung dieser Situation bietet Streaming-NNTP, indem hier am Anfang der Verbindung der Gegenseite einige Artikel angeboten werden. Mit jedem Artikel, der versandt wird, wird ein neuer angeboten und mit jeder

[14] Diese Messung wurde nur einmal durchgeführt, da das Durchsuchen aller Artikel im Dateisystem sehr aufwendig ist.

Bestätigung, daß ein Artikel angekommen ist, wird mitgeschickt, ob ein weiterer angebotener Artikel gesendet werden soll oder nicht. Sind alle angebotenen Artikel bearbeitet, schickt der Sender eine weitere Liste von Artikeln, die er noch anbieten möchte, bis alle Artikel übertragen sind. Hierdurch fließt die RTT für die Übertragung eines Artikels nicht mehr so stark in die gesamte Übertragung ein. In der Praxis wird mit Streaming-NNTP nahezu eine Verdoppelung der Übertragungsrate gegenüber herkömmlichen NNTP-Transfers erreicht.

```
1)  > check <19970722155700.LAA25140@ladder02.news.aol.com>
2)  > check <19970722155700.LAA25143@ladder02.news.aol.com>
3)  < 238 <19970722155700.LAA25140@ladder02.news.aol.com>
4)  > takethis <19970722155700.LAA25140@ladder02.news.aol.com>
5)  > [ article 878 ]
6)  > .
7)  < 238 <19970722155700.LAA25143@ladder02.news.aol.com>
```

Abbildung 15: Ausschnitt aus einem Streaming-NNTP-Transfer

Im Beispiel in Abbildung 15 werden in den Zeilen 1 und 2 dem Empfänger Artikel angeboten. In Zeile 3 bestätigt der Empfänger den ersten (bei ihm nicht vorhandenen) Artikel und teilt dem Sender mit, daß er doch gerne den Artikel aus Zeile 1 hätte. Dieser wird in den Zeilen 4 bis 6 übertragen. Mit der Bestätigung dieses Artikels in Zeile 7 wird dann auch gleich der nächste Artikel angefordert wird (aus Zeile 2). Ist die Liste, der durch “check” Kommandos angebotenen Artikel abgearbeitet, und der Sender hat noch Artikel vorrätig, so werden diese angeboten, wie oben beschrieben.

2.3.4 Schnelle NNTP-Transfers

Bei der Standardimplementierung von NNTP schreibt der NTA die Message-ID Header und Pfadnamen der Artikel, die ein Partnersystem haben möchte in eine Datei. Diese wird periodisch von einem Programm gelesen und die Artikel dann via NNTP verteilt (Abbildung 16).

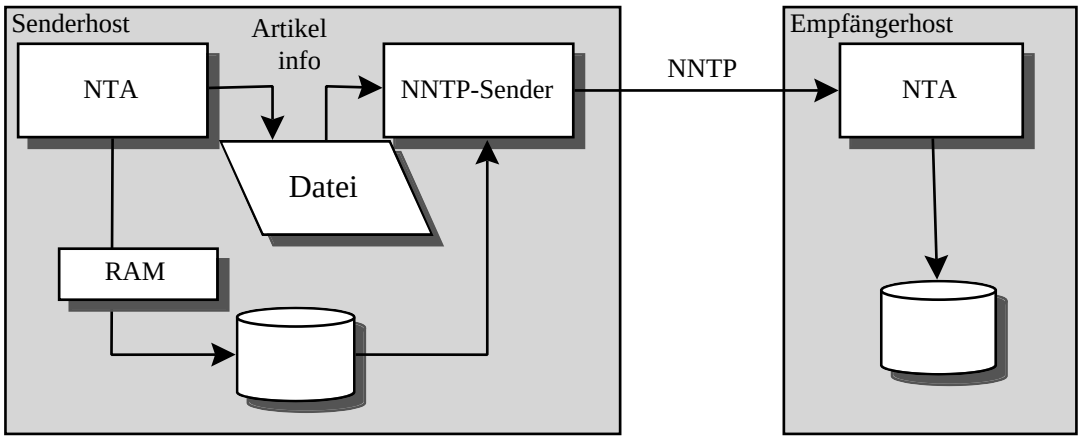


Abbildung 16: Verteilung mit herkömmlichem NNTP

Wenn das Sendeprogramm nicht innerhalb einer relativ kurzen Zeitspanne, nach der ein Artikel auf dem System angekommen ist, aufgerufen wird, stehen die Artikeldaten nicht mehr im Arbeitsspeicher des Rechners, sondern müssen erst im Dateisystem gesucht werden, was eine gewisse Zeit benötigt.

David Alden[15] hat mit nntplink ein Versendeprogramm entwickelt, das die Möglichkeit bietet, die Information über Artikel, die verschickt werden sollen, direkt über die Standardeingabe zu erhalten und so den Umweg über eine Datei zu ersparen, wie dies Abbildung 17 zeigt.

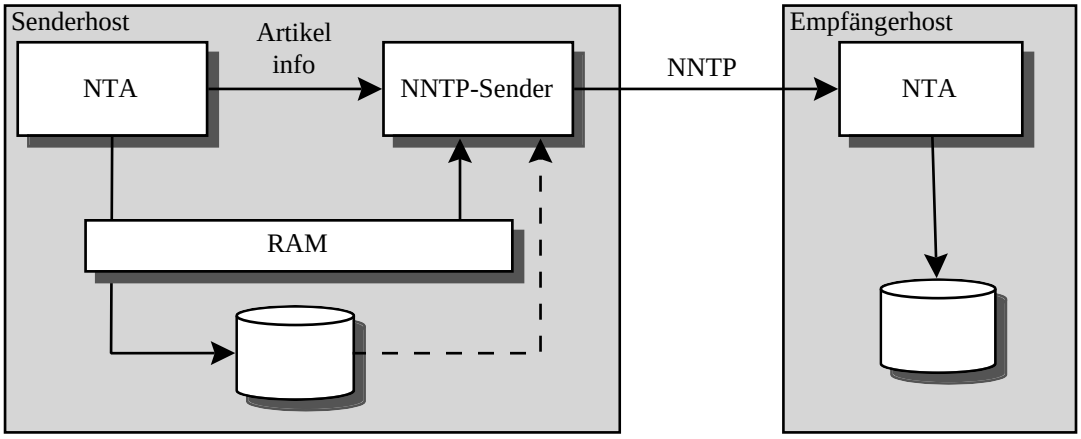


Abbildung 17: Verteilung mit NNTPlink

Hierbei wird versucht die Tatsache auszunutzen, daß die meisten Systeme im Hauptspeicher einen Plattenpuffer haben, in dem der Artikel noch eine Zeit lang vorhanden ist. In der Praxis funktioniert dies sehr gut und so werden die Artikel via nntplink sofort an die potentiellen Empfängersysteme weitergeleitet.

James Brister[16] hat mit innfeed ein Übertragungsprogramm entwickelt, das die Vorteile von Streaming-NNTP und nntplink vereint. Dank der Möglichkeit, mehrere NNTP-Verbindungen gleichzeitig zu bedienen, ist Innfeed sehr schnell und wird aktuell zwischen den großen Backboneservern bevorzugt eingesetzt.

2.4 Digitale Signaturen

Digitale Signaturen sind eine Möglichkeit, Daten so zu sichern, daß ihre Unverfälschtheit belegt werden kann. Dabei gibt es zwei Möglichkeiten, diese Signatur vorzunehmen:

- Die komplette Nachricht wird mit dem privaten Schlüssel des Senders verschlüsselt. Dies ist sehr rechenintensiv.
- Es wird nur ein Fingerabdruck der Nachricht, der i.A. kleiner ist, als die Nachricht selbst, mit dem privaten Schlüssel den Senders verschlüsselt.

Im Folgenden werden die involvierten Mechanismen im Einzelnen vorgestellt.

2.4.1 Message Digests

Message Digests *MD* sind Fingerabdrücke von Nachrichten, bei denen mit Hilfe von einfach berechenbaren Funktionen ein Wert ermittelt wird, der kürzer als die Originalnachricht ist. Die verwendete Funktion muß so beschaffen

[15] alden@math.ohio-state.edu
[16] brister@vix.com

sein, daß es relativ schwierig ist, eine zweite Nachricht zu generieren, die den gleichen Fingerabdruck hat.

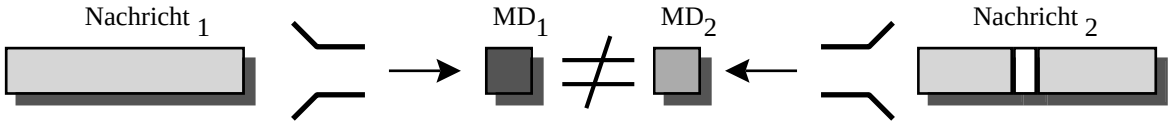


Abbildung 18: Message Digests zweier unterschiedlicher Nachrichten

In der Abbildung sind Nachricht_1 und Nachricht_2 nur leicht verschieden, dennoch liefert die Funktion stark unterschiedliche Werte MD_1 und MD_2. Eine (Hash-)Funktion, die die Eigenschaft hat, daß es sehr lange dauert, eine Nachricht n zu finden mit *hash(m) = hash(n)*, heißt *kryptographische Hashfunktion*.

Beispiele für solche kryptographischen Hashfunktionen sind Message Digest 2,4 und 5 (MD2 [Kal92], MD4 [Riv92a], MD5 [Riv92b]), Secure Hash Standard (SHS) oder RIPEMD-160 [DBP96]

2.4.2 Public Key Kryptographie mit RSA

Bei der Public Key Kryptographie besitzen Sender und Empfänger nicht denselben Schlüssel (wie bei symmetrischen Verschlüsselungsverfahren), sondern ein Schlüssel ist öffentlich und der zweite nur dem Besitzer bekannt. Diese Vorgehensweise wurde zum ersten Mal von W.Diffie und M.Hellman in [WD76] beschrieben.

1978 schlugen Rivest, Schamir und Adelman ein Public-Key Verschlüsselungsverfahren vor [Riv78], das auf der (von den meisten anerkannten) Annahme beruht, daß die Faktorisierung von Zahlen der “Länge” 100 Ziffern und mehr im allgemeinen schwierig ist und sehr lange dauert. Bei RSA gibt es zwei Schlüssel: den öffentlichen Key *e* und den geheimen Schlüssel *d*. Damit lässt sich RSA sowohl zur Verschlüsselung einsetzen (d.h., nur der Empfänger, der über den geheimen Schlüssel *d* verfügt, kann die Nachricht lesen, die mit *e* generiert wurde), als auch zur Authentisierung (jeder kann mit *e* verifizieren, daß eine Nachricht nur vom Besitzer von *d* stammen kann) verwenden.

2.4.3 Digitale Signaturen II

Um nun eine digitale Signatur einer Nachricht, *N* zu produzieren, wird von *N* zuerst ein Message-Digest

$$md = hash(N)$$

erzeugt, welcher dann mit dem privaten Schlüssel des Senders, *d* verschlüsselt wird. Im allgemeinen Fall wird diese Signatur *S* an die Nachricht angehängt und mit ihr verteilt.

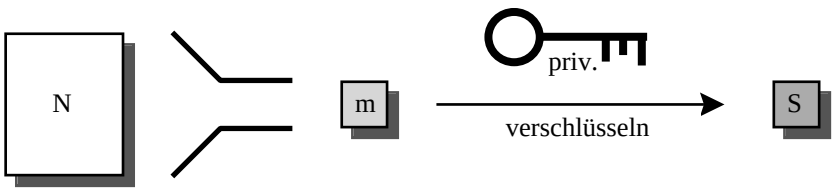


Abbildung 19: Eine digitale Signatur erzeugen

Ein Empfänger der Nachricht, der im Besitz des öffentlichen Schlüssels *e* ist, kann nun feststellen, daß die Nachricht zweifelsfrei von dem Absender kommt, von dem auch der Schlüssel *e* stammt. Desweiteren kann er fest-

stellen, daß die Nachricht *N* von diesem Absender stammt und nicht verändert wurde, wenn er ebenfalls einen Message-Digest von *N* bildet und dieser dann mit der entschlüsselten Signatur übereinstimmt.

2.5 Standards im Internet

Das Internet ist nicht zuletzt dadurch zu dem weltumspannenden Netzwerk geworden, daß die Standards, die den Betrieb ermöglichen, keine proprietären Standards sind, sondern in einem öffentlichen Diskussionsprozeß von der Internetgemeinde festgelegt werden. Die Standards im Internet werden als RFC (Request for Comment – Aufforderung zur Kommentarabgabe) bezeichnet. Das Dokument, in dem formuliert ist, wie ein RFC auszusehen hat, ist selbst ein RFC (RFC 1543 [Pos93]). Obwohl die RFCs im allgemeinen Sprachgebrauch als Standards bezeichnet werden, sind nicht alle RFCs wirklich Vorgaben, an die sich Benutzer und Entwickler halten müssen; es gibt noch Dokumente, die rein informellen Charakter haben (z.B. Verzeichnis der bisher erschienenen RFCs) oder Vorgehensweisen in bei bestimmten Dingen beschrieben (der Standardisierungsprozeß selbst ist mit RFC2026 [Bra96] solch ein RFC, der das Vorgehen beschreibt).

Der Standardisierungsprozeß läuft wie folgt ab:

1. Ein Autor reicht ein Dokument als sogenannter “Internet-Draft” ein.
2. Dieser “Internet-Draft” wird auf der Mailingliste <ietf-announce@ietf.org> bekanntgegeben und es wird zu Kommentaren aufgerufen.
3. Wenn der Draft innerhalb von sechs Monaten nicht aktualisiert oder von der IETF (Internet Engineering Task Force) als ausgereift genug betrachtet wird, wird er verworfen.
4. Wird der Draft als ausgereift genug betrachtet, wird er als “Experimental Standard” oder “Proposed Standard” in den Status eines RFCs gehoben.
5. Wenn es mindestens zwei unabhängige Implementierungen gibt, die zueinander kompatibel sind, kann das Dokument zu einem “Draft Standard” werden.
6. Wenn alle bekannten Probleme beseitigt sind, kann aus dem “Draft Standard” ein “Internet Standard” werden. Internet Standards werden noch einmal gesondert in der Reihe der STD Publikationen aufgeführt, wenngleich sie trotzdem weiterhin RFCs bleiben.

Der vollständige Standardisierungsprozeß wurde in RFC2026 [Bra96] veröffentlicht. Ein Verzeichnis der bisher veröffentlichten Internet-Drafts und RFCs findet man unter http://www.ietf.org/

2.6 Nummernvergabe im Internet

Im Internet müssen die verschiedenen Protokollnummern oder auch die Nummern der bekannten Ports eindeutig sein, um einen zuverlässigen Betrieb zu garantieren. Die *Internet Assigned Numbers Authority* (IANA) kümmert sich hierum. Um eine reservierte Portnummer oder eine reservierte Multicastgruppe zu erhalten, reicht es meist aus, eine EMail an iana@iana.org zu senden, in der der Zweck für die Nummer erklärt wird. Nach Prüfung durch die IANA wird die Nummer dann innerhalb von ca. vier Wochen zugewiesen. Listen der zugewiesenen Adressen findet man unter http://www.iana.org/

Bei manchen Adressen oder Nummern hat die IANA die Vergabe selbst wieder weiterdelegiert: Für das System der Domainnamen gibt es in den Ländern eigene Einrichtungen, die die Hoheit für eine Domain von der IANA erhalten haben und die innerhalb dieser Domain Unterdomains in Eigenregie vergeben können.

2.7 Zusammenfassung

In diesem Kapitel wurden Grundlagen beschrieben und Begriffe eingeführt, die für die nächsten Kapitel benötigt werden. Als erstes wurde die Internetprotokollfamilie mit ihren Erweiterungen für IP-Multicast vorgestellt. Danach wurden die NetNews als eine Anwendung präsentiert, die TCP/IP als Transportprotokoll nutzt. Als nächstes wurden Digitale Signaturen vorgestellt, mit denen sich die Echtheit und Unverfälschtheit eines Dokuments belegen lassen. Als Abschluß wurde der Prozeß der Standardisierung im Internet und die Vergabe von Protokollnummern erläutert.

3 Konzept

Wie bereits im vorherigen Kapitel erläutert sind Newsartikel Datagramme, die individuell transportiert werden. Mit IP-Multicast steht eine Möglichkeit bereit, Datagramme effizient an viele Empfänger zu transportieren, da die Artikel jeden Link nur einmal überqueren. Wie in Abschnitt 2.2 beschrieben, kann jedoch keine NNTP-Verbindung direkt über IP-Multicast geführt werden, so daß hierfür ein neues Protokoll geschaffen werden muß. Hier liegt es nun nahe, die Verteilung der Artikel von dem verbindungsorientierten Dienst NNTP, zu einem verbindungslosen zu verändern, bei dem dann einzelne Artikel als Datagramme transportiert werden.

3.1 Angestrebte Protokolleigenschaften

Nachfolgend wird nun eine Liste der Eigenschaften aufgezählt, die das zu entwickelnde Transportprotokoll erfüllen sollte.

Plattformunabhängig: Das Protokoll sollte nach Möglichkeit breit einsetzbar sein: Es soll nicht nur auf die Unix-Plattform beschränkt sein oder nur mit einem bestimmten Newstransportprogramm benutzbar sein.

8-Bit clean: RFC 1036 [HA87] spezifiziert in Bezug auf RFC 822 [Cro82], daß die in Newsartikeln gültigen Zeichen nur die des 7-Bit ASCII Zeichensatzes sind. Diese Beschränkung wurde bis heute nicht aufgehoben, gilt aber im (nicht amerikanischen) Usenet als veraltet und überholt.

Zwar steht mit MIME[FM96, NF97] eine Möglichkeit zur Verfügung nationale Sonderzeichen wie die deutschen Umlaute, so zu kodieren, daß RFC 822 befolgt wird. Dies wird von den Artikelschreibern meist nicht angewandt, da noch nicht genügend Newsuseragents zur Verfügung stehen, die MIME unterstützen. Hier in Deutschland ist der ISO-8859-1 Zeichensatz sehr populär und wird auch entsprechend oft verwendet. Damit gelangen aber Zeichen in Umlauf, die alle 8-Bits eines Bytes ausnutzen.

Das Transportprotokoll darf keine Verstümmelung dieser Zeichen zulassen, auch wenn es aktuell ein Verstoß gegen RFC 822 ist. Der Nachfolger von RFC1036, den Henry Spencer[17] derzeit vorbereitet[18], hebt die Beschränkung auf 7 Bit auf, so daß das Vorgehen dann legal und auch heute schon praktisch ist. Nebenbei: bei vielen anderen Anwendungsprotokollen gibt es die Beschränkung auf 7 Bit nicht, so daß die Vorgehensweise auch hinsichtlich der möglichen Wiederverwendung des Protokolls oder von Modulen in anderen Projekten sinnvoll ist.

Sparsam bezüglich Bandbreite: Das Volumen der Netnews ist relativ hoch. Bei einer durschnittlichen Größe der Artikel von ca. 4kByte (siehe auch 2.3.1) und ca. 350.000 Artikeln am Tag müssen mindestens 1500 MB am Tag[19] transportiert werden (bei manchen Rechnern, die auch noch einige lokale Hierarchien transportieren, kann das Volumen deutlich höher sein); dies entspricht einer konstanten Datenrate von mehr als 142 kBit/Sekunde (zum Vergleich: Ein ISDN-B-Kanal hat eine Bandbreite von 64kBit/s). Um die Datenrate niedriger zu halten, empfiehlt es sich die Daten vor dem Transport zu komprimieren, um damit Leitungsbandbreite zu sparen. Dabei ist zu beachten, daß die benötigte Rechenzeit für das (De-)Komprimieren möglichst gering ist, da sonst zu viel Zeit hierfür verwendet wird und der Gesamtdurchsatz zu stark sinkt.

[17] einer der Autoren von C News
[18] inzwischen wurde eine IETF-Workinggroup gegründet, die sich mit dieser Thematik beschäftigt
[19] Es gibt Administratoren auf im Usenet, die heute schon von bis zu 6GB am Tag reden ...

Ausfalltolerant: Sowohl Netzausfälle, als auch Ausfälle der Serverrechner sind an der Tagesordnung. Das Usenet ist weitgehend gegen Ausfälle immun, da Artikel dann über Umwege zu den nächsten Servern gelangen. Wenn allerdings ein Endsystem oder die Strecke zum Endsystem ausfällt, müssen Vorkehrungen getroffen werden, daß die Artikel, die nicht am Ziel angekommen sind, nach Behebung der Fehlersituation nachträglich zugestellt werden. Es sind die folgenden Fälle zu unterscheiden:

- Leitungsausfall: Da der Router vom nächsten Router keine Informationen mehr erhält, wird der Zweig hinter der Störungsstelle aus der Forwardingtabelle genommen. Die Datagramme, die zu dem Zeitpunkt verschickt werden, können nicht beim Empfänger ankommen.
- Ausfall Endsystem: Hier gilt im Prinzip dasselbe wie bei Leitungsausfall. Wenn jedoch in der "Nähe" des Endsystems ein zweiter Empfänger ist, kann dieser die Daten entgegennehmen und sie dem ausgefallenen Endsystem nach dessen Wiederinbetriebnahme zustellen.
- Ausfall Empfänger NTA: Wenn nur der Empfänger NTA ausfällt, können trotzdem weiterhin Datagramme empfangen werden. Diese Datagramme sollten der Effizienz wegen, zwischengespeichert und dem NTA später übergeben werden.
- Paketverlust auf der Leitung: Wird kein "Zuverlässiges Multicast Protokoll" (siehe 2.2.1) gewählt, können durch Paketverluste auf der Leitung Datagramme und auch Artikel verloren gehen.

In allen vier Fällen kann es also passieren, daß Datagramme nicht beim Empfänger NTA ankommen und deswegen Artikel verloren gehen.

Transportsicherheit: Da Router im allgemeinen nicht mitprotokollieren, welche Pakete von wo nach wo geflossen sind, ist es relativ einfach, einen Artikel in das System zu bringen, ohne daß nachvollziehbar ist, von wem er stammt. Somit können, wenn keine Vorkehrungen getroffen werden, relativ einfach gefälschte Artikel oder Werbeartikel in dem Umlauf gebracht werden, ohne daß der Sender ermittelbar ist. Es sollten deshalb Maßnahmen entwickelt werden, dies zu verhindern.

Geschwindigkeit des Versands: Neben der Bandbreitenersparnis ist die Geschwindigkeit, mit der die Artikel verteilt werden ein weiteres wichtiges Kriterium. Ein Vergleich aus dem Strassenverkehr zeigt dies deutlich: das Zwei-Liter Auto wird nicht akzeptiert, wenn man damit nur 30 km/h schnell fahren kann.

Es sollte also versucht werden, an die in Abschnitt 2.3.3 auf Seite 17 ermittelten Artikellaufzeiten heranzukommen, damit die Software wirkungsvoll eingesetzt werden kann – andernfalls sind die Artikel bereits via NNTP auf dem Zielsystem angelangt und der Nutzen von *Mcntp* wird in das Gegenteil gekehrt.

Da die Benutzer sich daran gewöhnt haben, daß ihre Postings in sekundenschnelle über den Erdball verteilt werden, und sie schon wenige Minuten später eine Antwort auf ihre Fragen erhalten, ist es nicht praktikabel, die Verteilung zu verzögern. Diese bedeutet, daß die Verteilung via *Mcntp* genau so schnell sein sollte, wie über andere Transportmechanismen.

Benötigte Rechenzeit: Datenkompression und -verschlüsselung benötigen Rechenzeit. Von dieser Zeit ist direkt die Rate abhängig, mit der Artikel versandt werden können. Bei 300.000 Artikel am Tag darf die Verarbeitung eines Artikels höchstens eine viertel Sekunde benötigen, um mit der Artikelanzahl mithalten zu können.

3.2 Designentscheidung: Welches Multicastprotokoll?

Die Entscheidung, welches Multicastprotokoll verwendet werden soll, fällt aus den folgenden Gründen auf das normale IP-Multicast, wie es in RFC1112 [Dee89] definiert wurde:

- Fast alle Router unterstützen heutzutage IP-Multicast nach RFC1112 und mit dem MBone existiert bereits ein (virtuelles) Netzwerk, welches diese Technologie unterstützt.
- Fast alle modernen Betriebssysteme unterstützen Multicast nach RFC1112, so daß hier kein zusätzlicher Aufwand getrieben werden muß.
- Die Reihenfolge, in der die Artikel auf einem Zielsystem eintreffen ist unwichtig. Eine Reihenfolge der Ankunft der Artikel ist auch bei sonstigen Transportmechanismen nicht garantiert. [20]
- Mit NNTP steht ein erprobter Mechanismus zur Verfügung, um eventuelle Löcher in der Übertragung zu schließen. Wenn die Artikel schon auf dem Zielsystem vorhanden sind, beschränkt sich die NNTP-Übertragung auf *"ihave" – "435 Duplicate"*.
- Reliable Multicast benötigt zusätzliche Hard- oder Software; für die Systemkerne von kommerziellen Systemen, die ohne Quellcode ausgeliefert werden, ist es fast unmöglich, diese Erweiterungen im Kern vorzunehmen.
- Es gibt viele verschiedene Ansätze für Reliable Multicast, aber keiner scheint komplett ohne Probleme zu sein.

3.3 Ein erster Ansatz

Ein erster einfacher Ansatz sieht wie folgt aus:

- Ein Sender *S* nimmt alle Artikel, die auf dem Host eintreffen, packt sie in Datagramme und schickt sie an eine vorher festgelegte Multicastgruppe.
- Ein Empfänger *E* wird Mitglied dieser bekannten Multicastgruppe. Wenn ein Paket eintrifft, wird der Inhalt an einen lokalen Newsserver übergeben.

Diese Vorgehensweise ist performant, hat aber auch die folgenden Nachteile:

- Alle Artikel werden in einem Datenstrom gesendet. Die Empfänger erhalten auch Newsgruppen bzw. -hierarchien, die sie nicht bekommen wollen; erst nach Parsen des Artikelkopfes kann entscheiden werden, ob der Artikel gewünscht wurde oder nicht.
- Sollen Hierarchien mitverteilt werden, die nur lokale Bedeutung haben, so werden diese entweder zu weit verteilt oder, wenn die Ausbreitung beschränkt wird, die Gruppen mit überregionaler Bedeutung nur lokal verteilt.

[20] Außerdem sortieren Newsuseragents, NUAs die Artikel in der Reihenfolge, in der sie der Leser haben möchte. Diese Reihenfolge muß nicht unbedingt mit der übereinstimmen, in der die Artikel auf dem Server angekommen sind, oder gar in das Usenet eingespeist wurden.

Eine Verbesserung ergibt sich, wenn für jede Newshierarchie oder -gruppe eine eigene Multicastgruppe gewählt wird. Hier muß dann entweder der Sender die Artikel je nach Newsgruppe in unterschiedlichen Multicastgruppen senden oder es muß pro Hierarchie ein Sender vorhanden sein.

Um eine weitere Verbesserung zu erzielen, sollte die Zuordnung dynamisch sein. Außerdem sollten die Daten vor dem Versand nach Möglichkeit komprimiert werden, um die benötigte Leitungsbandbreite zu reduzieren. Weiterhin müssen die Pakete noch digital signiert werden, um Verfälschungen während des Transports oder unberechtigt in den Transport eingebrachte Pakete erkennen zu können. In den nächsten Abschnitten wird dies näher behandelt.

3.4 Zuordnung News- zu Multicastgruppen

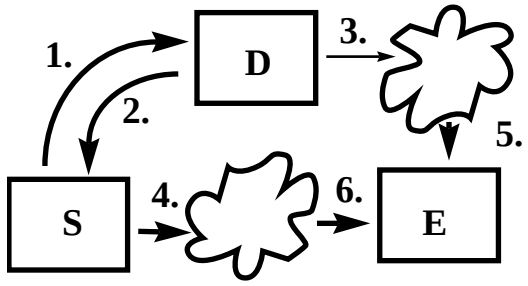
Wie bereits im vorherigen Abschnitt geschrieben, sollte die Zuordnung von Newsgruppen und -hierarchien zu Multicastgruppen möglichst dynamisch erfolgen:

- Multicastgruppen können mehrfach genutzt werden, wenn lokale Hierarchien nur lokal verteilt werden.
- Es sind beliebige Abstufungen in der Stückelung der Newsgruppen oder -hierarchien möglich.
- Es muß nur eine Multicastgruppe fest vergeben und weltweit bekannt sein (für die Zuordnung von Newsgruppen und -hierarchien zu Multicastgruppen).

3.4.1 Directory Server

Um dies zu erreichen wird ein Directory Server *D* eingeführt, der diese Zuordnung koordiniert. Der Directory Server vergibt auf Anfragen hin Multicastgruppen und teilt diese den Empfängern mit. Diese sind damit in der Lage, die passenden Gruppen für sich zu abonnieren. Der prinzipielle Ablauf wird wie folgt aussehen:

1. Der Sender baut eine Verbindung zum Directory Server auf und teilt ihm die Newsgruppe mit, die er versenden möchte.
2. Der Directoryserver teilt dem Sender eine passende Multicastgruppe mit.
3. Der Directoryserver annonciert die Gruppe
4. Der Sender beginnt die Artikel zu senden
5. Der Empfänger hört auf Announcements
6. Wenn dem Empfänger eine Gruppe gefällt, abonniert er diese und erhält die Artikel



S: Sender E: Empfänger
D: Directoryserver

Abbildung 20: Directory Service

Die Ankündigungen des Directory Servers werden selbst in einer speziell dafür vorgesehenen Multicastgruppe verteilt und regelmäßig wiederholt, so daß auch Empfänger, die „erst später eingeschaltet haben“, die Ankündigungen mitbekommen. Diese Multicastgruppe sollte eine von der IANA (siehe Abschnitt 2.6 auf Seite 22) zugewiesen Gruppe sein. Die Benutzung einer Multicastgruppe für Ankündigungspakete hat den Vorteil, daß diese Lösung viel besser skaliert, als wenn die Empfänger eine Verbindung zum Directory Server öffnen müßten (oder gar die Gegenrichtung), oder wenn die Pakete via Unicast verteilt würden.

Zur Benutzung mit *Mcntp* wurde von der IANA bereits die Multicastgruppe MCNTP-DIRECTORY.MCAST.NET mit der Class-D Adresse 224.0.1.51 offiziell reserviert.

3.4.2 Plazierung der Directory Server

Die einfachste Möglichkeit für die Plazierung wäre einen einzigen Server an einer zentralen Stelle aufzustellen, der für alle News Sender gut erreichbar wäre. Wenn ein Server News senden möchte, baut er eine Verbindung zu diesem Server auf und bekommt eine Gruppennummer zugeteilt. Während dies noch für kleine Netzwerke praktikabel ist, skaliert es im globalen Internet sehr schlecht und ist außerdem nicht fehlertolerant.

Die Idee ist deshalb, diese Directory Server ebenfalls zu verteilen – jeder Senderhost bekommt einen Server, den die lokalen Sender sehr schnell befragen können. Die Synchronisation der Directory Server läuft so ab, daß ein neuer Server die Gruppe MCNTP-DIRECTORY.MCAST.NET betritt und eine gewisse Zeit wartet. Erhält er ein Ankündigungspaket, nimmt er die Gruppen mit in seine Tabellen auf. Wird während der Wartezeit keine Ankündigung erhalten, kann er sich als alleine im Netz betrachten und die Gruppen frei wählen. Die nächste Grafik zeigt, wie dies dann aussieht.

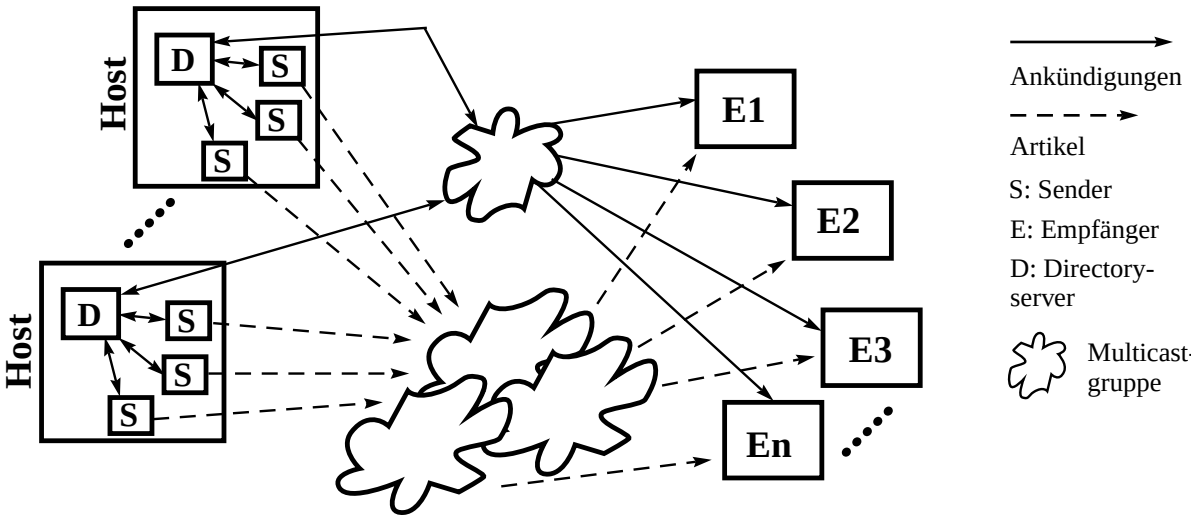


Abbildung 21: Skalierung des Directory Services

3.5 Datenkompression

Um die benötigte Leitungsbandbreite weiter zu reduzieren, sollen die Pakete, bevor sie in's Netz geschickt werden, komprimiert werden. Hierbei ist zu beachten, daß jedes einzelne Paket für sich komprimiert werden muß, da ansonsten bei auftretenden Paketverlusten oder bei Vertauschungen in der Reihenfolge, in der die Artikel beim Empfänger ankommen, alle Artikel, die nach dem Fehlenden oder Vertauschten eintreffen, nicht mehr dekomprimiert werden können und damit verloren sind.

Für die Datenkompression ist eine Huffmancodierung[Sed88] geeignet, da sowohl das Komprimieren, als auch das Dekomprimieren mit relativ wenig Aufwand erfolgen kann. Das Hauptproblem ist die Generierung des Huffman-Baumes; dies kann aber offline erfolgen und wenn es größere Änderungen gibt, wird dieser geänderte Baum an die Empfänger verteilt, um die Änderungen nachzuvollziehen.

Ein Problem hierbei könnte sein, daß der Baum je nach Hierarchie oder Sprache unterschiedlich aussieht – so ist die Verteilung der Zeichen in der deutschen Sprache anders als z.B. in der französischen. Für die Gruppen in denen viele Binärdateien gepostet werden, wird der Baum ganz anders aussehen, da die Binärdateien meist via uuencode(1) oder Base64 [BF93] kodiert sind. Um dieses Problem zu mildern, gibt es zwei Möglichkeiten:

- Für unterschiedliche Hierarchien werden verschiedene Bäume generiert und genutzt. Hier steigt der Aufwand der Pflege mit der Anzahl der Bäume.
- Einen Baum aus einer großen Anzahl von Artikeln benutzen, welche aus allen Hierarchien stammen. Hier sind zwar die Ergebnisse schlechter, als oben, dafür ist der Aufwand deutlich geringer.

Laut [Sed88] erreicht man mit der Huffman-Kompression ca. 20% Datenreduktion auf Englischen Text. Dies ist weniger als bei der Kompression mit compress oder gzip; die Reduktion ist jedoch deutlich billiger zu erhalten und es können mit den Datagrammen keine Tabellenänderungen verloren gehen.

Eine andere Alternative, die sich trotz kleiner zu komprimierender Dateien anbietet, ist die Verwendung der zlib, wie sie in den RFCs 1950-52 [LD96, Deu96a, Deu96b] beschrieben wird. Dabei wird jeder Artikel einzeln komprimiert und trägt bei der Übertragung seine Codetabelle bei sich, so daß er auch im Fall von Paketverlusten oder vertauschter Reihenfolge erfolgreich dekodiert werden kann. Der Nachteil, daß zlib mehr Rechenzeit benötigt, als die Verwendung des Huffman-Algorithmus, wird durch die ca. doppelt so hohe Datenreduktion wieder wettgemacht. Außerdem fällt die Notwendigkeit weg, Kodierungstabellen zu pflegen.

3.6 Datensicherheit

Bei den NetNews ist im allgemeinen Fall das Ausspähen der Daten weniger ein Problem (interne oder kostenpflichtige Gruppen einmal ausgenommen). Ein großes Problem ist vielmehr die Möglichkeit, Artikel mit gesetzeswidrigem Inhalt oder gefälschte Kontrollnachrichten (siehe auch Abschnitt 2.3 auf Seite 12) in das Usenet einzuspeisen, ohne Spuren zu hinterlassen. So hat vor einigen Jahren ein Benutzer die Wirkung von *sendsys* Kontrollnachrichten lokal testen wollen und hat dabei nicht aufgepasst. Sechs Nachrichten haben sein System verlassen und wurden weltweit verbreitet. Dies hatte zur Folge, daß zigtausende Systeme weltweit ihm eine Antwortmail senden wollten. Die zuständigen Mailgateways waren tagelang verstopft.

Beim herkömmlichen Transport der Artikel via NNTP oder UUCP hinterlassen die Systeme, durch die der Artikel läuft im *Path:* Header ihre Kennung. Zudem schreiben sie die *Message-ID* des Artikels zusammen mit dem *Path:* Headers des Hosts, von dem sie den Artikel erhalten haben in eine Datei. Wenn nun Probleme auftreten, können die Administratoren der Hosts, die auf dem Pfad liegen, auf dem der Artikel gewandert ist, versuchen dem Problem auf den Grund zu gehen. Dies gelingt zwar nicht immer, aber die Chancen stehen nicht schlecht, zumindest eine Wiederholung der Probleme zu verhindern.

Beim Transport von NetNews via IP-Multicast geht dies nicht mehr, da die Router auf dem Weg vom Sender zum Empfänger zwar zu Accountingzwecken mitloggen können, von wo nach wo wieviele Bytes geflossen sind, aber sonst keine nähere Auskunft über den Inhalt der Pakete geben. Ist gar die Absender IP-Nummer falsch, gibt es keine Möglichkeit nachzuverfolgen, wer einen Artikel abgeschickt hat.

Aus diesem Grund ist es nötig, daß die Empfänger sicher sein können, daß die Artikel von einem ihnen bekannten Sender kommen. Dieser Sender wird wie beim herkömmlichen Transport, alle Artikel, die das System passieren in den Logfiles vermerken, so daß Probleme verfolgt werden können. Um diese Sicherheit zu erreichen, werden digitale Signaturen (Abschnitt 2.4 auf Seite 20) verwendet, mit denen die Artikel vor dem Versand unterzeichnet werden.

Ein Punkt, der an dieser Stelle nicht unterschätzt werden darf, ist die Geschwindigkeit, mit der das Ver- und Entschlüsseln des Message Digests vonstatten geht. Während ein Artikel nur ein einziges Mal eingepackt wird, kann es sein, daß mehrere Tausend Empfänger diesen Artikel empfangen. Außerdem ist es nötig, um die gewünschte Bandbreitenersparnis in der Praxis zu erzielen, möglichst viele Artikel mittels *Mcntp* zu den Empfängern zu bringen.

3.7 Versand der Daten

Hier soll nocheinmal zusammenhängend dargestellt werden, was beim Versand eines Netnews-Artikels via *Mcntp* geschieht. Als erstes wird der Artikel auf Wunsch komprimiert. Dann wird von dem möglicherweise komprimierten Artikel ein Message-Digest gebildet, welcher dann mit dem privaten Schlüssel des Senders verschlüsselt wird, um so eine digitale Signatur zu erhalten. Diese Signatur wird mit dem möglicherweise komprimierten Artikel und einem Protokollkopf von *Mcntp* zu einem Paket zusammengepackt und dann als UDP-Paket versandt.

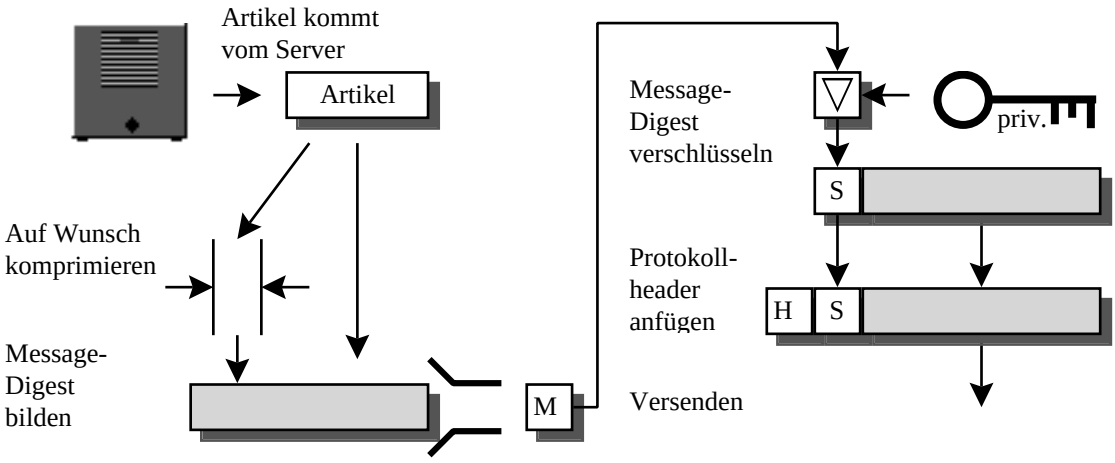


Abbildung 22: Artikel zum Versand fertig machen

Der Empfänger wird nach Erhalt des Paketes den Fingerabdruck des Senders mit dessen öffentlichem Schlüssel entschlüsseln und mit einem Fingerabdruck der Daten, den er selbst angefertigt hat, vergleichen. Stimmen diese beiden Fingerabdrücke überein, so liegt ein gültiger Artikel vor, der eventuell noch dekomprimiert werden muß und dann an das Newssystem weitergeleitet werden kann. Ist keine Übereinstimmung vorhanden, so ist der Artikel zu verwerfen.

3.8 Eine Bestandsaufnahme

Um das hinter *Mcntp* stehende Konzept besser beurteilen zu können, sollen hier noch einige mögliche Alternativen zu *Mcntp* mit ihren Vor- und Nachteilen dargestellt werden:

News via Satellit Der Versand von NetNews via Satellit bringt natürlich noch eine größere Bandbreitenersparnis, als *Mcntp* über terrestrische Leitungen, da die Artikel nicht terrestrisch übertragen werden. Prinzipiell gibt es zwei Ansätze für die Übertragung:

- Die Artikel werden mehrfach ausgestrahlt, so daß die Empfänger in der Lage sind, Artikel, die beim ersten Mal verloren gegangen sind, später zu erhalten.
- Die Artikel werden nur einmal ausgestrahlt und werden dann später nocheinmal terrestrisch angeboten, um Lücken zu füllen.

Der Versand via Satellit hat den Nachteil, daß zusätzliche – meist proprietäre – Hardware benötigt wird. Außerdem wird es sehr teuer, wenn man selbst einen Newsfeed via Satellit verteilen möchte; es ist deshalb nur möglich, Newsgruppen zu erhalten, die von allgemeinem Interesse sind.

Interessanterweise könnte sich *Mcntp* als Übertragungsprotokoll für den Versand via Satellit eignen, wenn als Netzwerkprotokoll auf der Satellitenstrecke IP benutzt wird.

Newsserver an den Knoten Zusätzliche Newsserver an den Netzwerkknoten, bei denen mehrere Leitungen sich verzweigen, können auch die Mehrfachübertragungen verhindern. So wird in Abbildung 23 die Strecke R1 – R2 von den Artikel nur noch einmal überquert, wenn bei Router R2 ein zusätzlicher Newsserver installiert wird, wie dies rechts dargestellt ist.

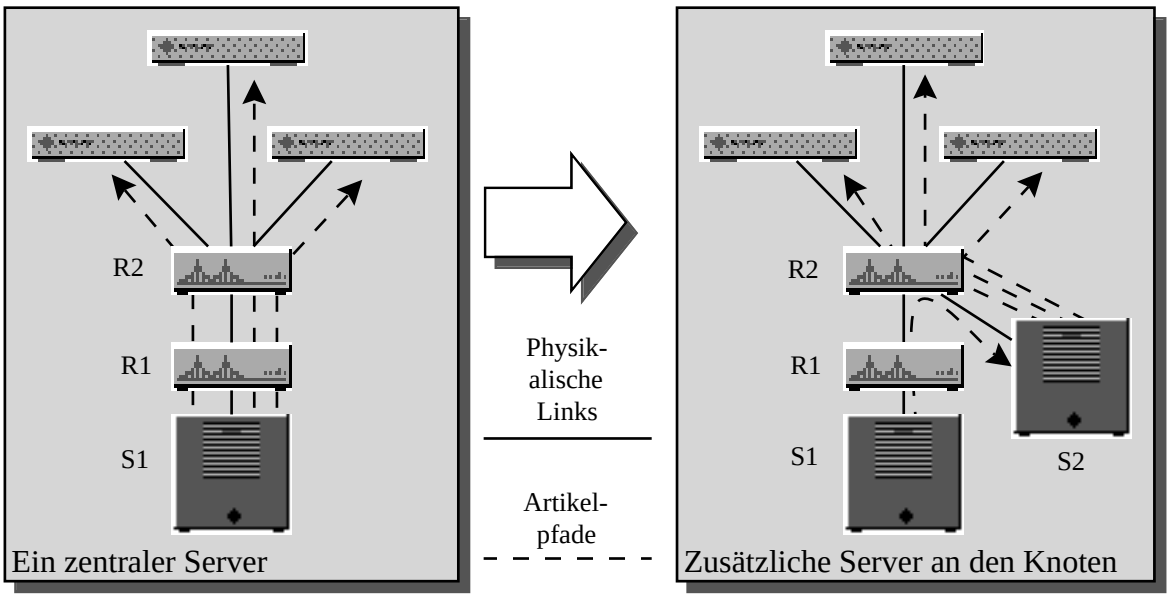


Abbildung 23: Zusätzliche Server an den Knoten

Die Vorteile dieser Vorgehensweise liegen neben der Bandbreitenersparnis darin, daß der Transport zwischen den Servern per NNTP geschehen kann und damit sicher und schnell ist.

Der deutliche Nachteil ist, daß die Anschaffung und die Administration zusätzlicher Newsserver relativ teuer ist: neben der Investition in Hardware (Rechner, Hauptspeicher, Festplatten), muß der Server betreut und gewartet werden.

News über TV-Netze In der Weise, wie Videotext heutzutage in der Austastlücke des Fernsehbildes übertragen wird, könnten statt dieser Tafeln auch Newsartikel übertragen werden. Dies könnte analog der Verteilung von Programmen geschehen, wie dies der “WDR Computerclub” [21] mit seinem Videotext Decoder schon vor Jahren realisiert hat, was sich aber nicht durchsetzen konnte. Ein Problem ist hier auch wieder die proprietäre Hardware und die auch relativ geringe Bandbreite.

[21] http://www.wdr.de/tv/Computer-Club/

Inzwischen gibt es in manchen Städten in der Bundesrepublik Versuche, über die Breitbandkabelnetze auch IP-Daten zu transportieren. Dabei werden die Daten zu den Nutzern anstelle von Fernsehbildern über das Kabelnetz übertragen. Für den Datentransport in der Rückrichtung muß der Benutzer weiterhin eine normale Verbindung über das Telefonnetz zu seinem Provider aufbauen.

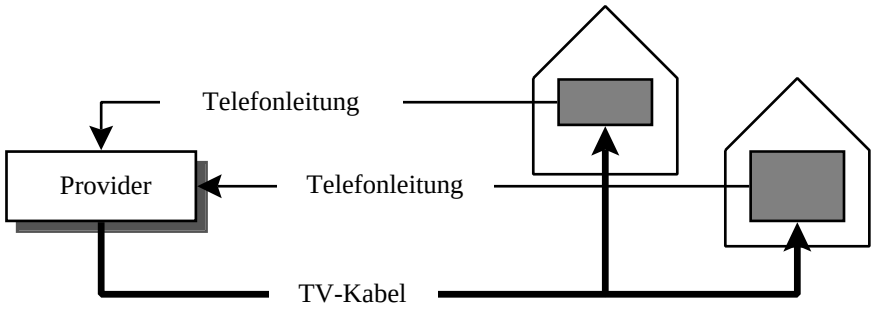


Abbildung 24: Newstransport über das Kabelnetz

Der klare Vorteil dieser Technik liegt darin, daß heute schon große Teile der Bevölkerung über einen Kabelanschluß verfügen. Außerdem hat der Vorwärtskanal eine hohe Bandbreite, was speziell für die Verteilung von NetNews interessant ist. Diese Technik ist allerdings noch nicht richtig ausgereift und Versuche, die stattfinden, gehen meist in die Richtung, noch mehr Shoppingkanäle anzubieten.

Wie beim Versand über Satellit könnte aber auch hier *Mcntp* als Transprotokoll zur Verteilung der Newsartikel eingesetzt werden.

3.9 Zusammenfassung

Mcntp als Transportprotokoll für die Verteilung von NetNews über IP-Multicast nutzt IP-Multicast so, wie es von Steve Deering in RFC1112 [Dee89] definiert wurde. Es wurde bewußt auf eine Anwendung von Reliable Multicast Protokollen verzichtet, da diese noch in den Kinderschuhen stecken und Modifikationen an Systemkernen oder zusätzliche Hardware benötigen, während Multicast nach RFC1112 bereits auf dem MBone in praktischen Einsatz ist. Darüberhinaus unterstützen fast alle modernen Betriebssysteme und Router bereits Multicast nach RFC1112.

Dadurch, daß Artikel nur einmal über eine Verbindung geschickt werden müssen, wenn auch mehrere Empfänger am Ende des Links diese erhalten möchten, ergibt sich eine große Einsparung an Leitungsbandbreite. Selbst, wenn auf dem Transport verlorengegangene Artikel mittels NNTP nachgeschickt werden, ist der gesamte Bandbreitenbedarf geringer, als wenn alle Artikel über NNTP verschickt werden müßten. Der Einsatz von Mechanismen zur Datenkompression vor dem Versand, reduziert dabei die benötigte Bandbreite zusätzlich.

Da die am Transport beteiligten Router keine Logdateien produzieren, aus denen hervorgeht, wo genau ein Newsartikel in das System eingespeist wurde, muß dies über andere Mechanismen geschehen. Hierfür werden alle Artikel vor dem Versand kryptographisch mittels eines privaten Schlüssel signiert. Die Empfänger akzeptieren nur Artikel, die eine gültige Signatur eines bekannten Absenders enthalten, was mit Hilfe dessen öffentlichen Schlüssels überprüft wird.

Die Zuordnung von Newsgruppen und -hierarchien zu den Multicastgruppen erfolgt dabei dynamisch durch einen speziellen Directroryserver, der die Gruppen an Sender verteilt und die Zuordnung dann über eine extra dafür von der IANA reservierte Multicastgruppe an die möglichen Empfänger verteilt.

Im Gegensatz zu Alternativen, wie dem Transport via Satellit oder über Kabelnetze, kann *Mcntp* die vorhandene Infrastruktur nutzen und benötigt keine zusätzliche Hardware, die oft proprietär ist.

4 Implementierung

Im Folgenden wird die Implementierung von *Mcntp* beschrieben. Die Funktionsweise von *Mcntp* folgt im Wesentlichen der Beschreibung, wie sie im Kapitel “Konzept” erarbeitet wurde. Nachfolgend kommt zuerst eine Übersicht über *Mcntp* und dann die Beschreibung der einzelnen Teile.

*Mcntp*wurde in der Programmiersprache “C” entwickelt.

4.1 Überblick

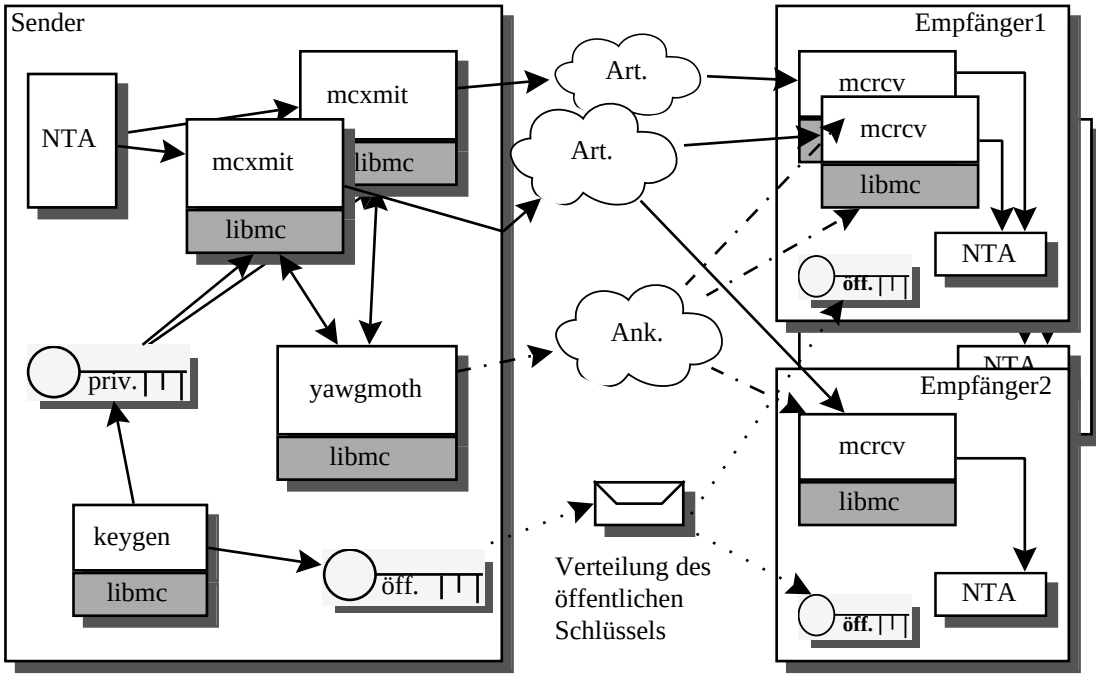


Abbildung 25: Übersicht über *Mcntp*

Mcntp besteht aus fünf Teilen:

- Die Multicastbibliothek, libmc – fast alle anderen Programme benutzen Routinen aus dieser Funktionenbibliothek.
- Der Directoryserver, yawgmoth[22] – koordiniert die Zuordnung von News- zu Multicastgruppen
- Der Multicastsender, mcxmit – versendet Artikel via IP-Multicast
- Der Multicastempfänger, mcrcv – empfängt die Artikel als Multicastpakete und leitet sie an Newsserver weiter

[22] Zuerst sollte der Directoryserver mcctld für “Multicast Control Dämon” heißen. Da dies einigen Leuten zu nichtssagend war wurde nach einem neuen Namen Ausschau gehalten. Als nächstes kam *Erwin* in das Gespräch, aber können Sie sich einen Dämonen namens Erwin vorstellen? Der jetzige Name *yawgmoth* stammt von einer Karte des Spiels *Magic the Gathering* (siehe auch http://www.wizards.com/) und ist die einzige Karte, die gefunden wurde, die einen Dämonen zeigt.

- Hilfs- und Demoprogramme – Zusätzliche Programme, die für den Betrieb von *Mcntp* nützlich oder notwendig sind

4.2 Die Funktionenbibliothek, libmc

Die Funktionenbibliothek libmc bietet dem Programmierer eine definierte Schnittstelle zu verschiedenen Funktionen, die bei der Programmierung von IP-Multicastanwendungen notwendig oder nützlich sind. Bei der Implementierung wurde Wert auf Modularität gelegt – so wurden alle gemeinsam genutzten Routinen in einer Bibliothek abgelegt, um sie so auch anderen Programmen zur Verfügung zu stellen. Hier soll nun eine Kurzbeschreibung der wichtigsten Routinen mit kurzen Beispielen folgen.

Die Funktionen der Bibliothek wurden nach Aufgabenbereichen gegliedert; vor den Funktionen wird deshalb der Aufgabenbereich und die entsprechende C Headerdatei angeben.

TCP Sockets – tcp_sock.h

Diese beiden Funktionen erleichtern das Öffnen von TCP-Sockets.

open_tcp_conn(char *host,int port) Diese Funktion öffnet eine TCP-Verbindung zum angegebenen Host `host` auf den angegebenen Port `port`. Gibt einen Dateideskriptor zurück, welcher dann mittels read() und write() gelesen und beschrieben werden kann. Beispiel:

```
int s;
char buf[100];
s=open_tcp_conn("news.some.domain",119);
if(s>=0) {
    /*  socket is offen */
    write(s,"Hello World",11);
    close(s);
}
```

open_tcp_serv() Diese Funktion öffnet einen TCP-Socket, der in der Lage ist, Verbindungen entgegenzunehmen. Damit kann dann relativ einfach ein Server realisiert werden; im folgenden Beispiel wird ein Server auf Port 12345 generiert der nur Verbindungen von localhost akzeptiert.

```
int s,f;
s=open_tcp_serv("localhost",12345");
   ....
f=accept(s,0,0);
   ....
```

Multicast Sockets – mc_sock.h

Diese Routinen vereinfachen die Arbeit mit den für die Multicastkommunikation erforderlichen Sockets. Die Funktionen send_to() und recv_from() haben als ersten Parameter einen Zeiger auf eine `mc_sock` Struktur. Diese Struktur wird von den Funktionen get_send_socket() und get_read_socket() initalisiert.

get_send_socket(u_short port,char *gruppe,u_short ttl) Diese Funktion öffnet einen Socket, der anschließend zum Versand von Paketen an die angegebene Multicastadresse `gruppe` und UDP-Port `port` genutzt werden kann. Die Pakete erhalten beim Versand eine TTL von `ttl`. Beispiel:

```
struct mc_sock *send_sock;
send_sock=get_send_socket(5432,"mcntp-directory.mcast.net",64);
```

get_read_socket(u_short port,char *gruppe) Diese Funktion öffnet einen Socket, der anschließend zum Empfang von Paketen der angegebenen Multicastgruppe `gruppe` auf dem angegebenen UDP-Port `port` genutzt werden kann. Beispiel:

```
struct mc_sock *read_sock;
read_send_sock=get_read_socket(5432,"mcntp-directory.mcast.net");
```

send_to(struct mc_sock *to,char *buf,int len) Versendet den Inhalt eines Puffers `buf` mit der Länge `len` als ein Paket über den angegebenen Socket `to`, der zuvor mit get_send_socket() initialisiert worden sein muß. Beispiel:

```
/* send_sock von oben */
res=send_to(send_sock,"Hello World",11);
if(res!=11)  {  /* Fehler */  }
```

recv_from(struct mc_sock *from,char **buf, int len,int mode) Liest ein Paket von dem angegebenen Socket `from` in den Puffer `*buf` ein. Es werden höchstens `len` Bytes gelesen. Wenn `mode` auf Null gesetzt ist, wartet recv_from() bis Daten zum Lesen vorhanden sind. Ist `mode` auf Eins gesetzt, so kehrt recv_from() sofort zurück, selbst wenn keine Daten vom Socket gelesen werden konnten. Beispiel:

```
/* read_sock von oben */
char buf[200];

res=read_from(read_sock,&buf,200,0);
printf("Es wurde folgender Text erhalten: %s\n",buf);
```

multicast_dont_loop(struct mc_sock *mcs) Normalerweise erhalten Multicastempfänger, die auf dem gleichen Host sind, wie der Sender auch die Daten vom Sender (vgl. 2.2.2). Mit dieser Funktion kann dieses Verhalten unterbunden werden.

String Routinen – mystring.h

Diese Routinen zur Behandlung von Zeichenketten fehlen sehr oft in den C Bibliotheken. Gerade die Routinen, die bei ihrer Arbeit nicht nach Groß-/Kleinschreibung unterscheiden, sind für *Mcntp* wichtig. Diese Routinen werden nicht detailliert beschrieben, da sie den, in der Standard C Bibliothek vorhandenen Routinen, sehr ähnlich sind und die Anwendung dort nachgeschlagen werden kann.

stricmp() Vergleicht zwei NULL (\0) terminierte Strings, wie strcmp(). Diese Funktion unterscheidet nicht nach Groß-/Kleinschreibung.

strnicmp() Vergleicht zwei NULL (\0) terminierte Strings, wir strncmp() jedoch nur bis zu einer bestimmten Länge. Diese Funktion unterscheidet nicht nach Groß-/Kleinschreibung.

stristr() Sucht die eine Zeichenkette in der anderen. Wird sie gefunden, wird ein Zeiger darauf zurückgegeben. Diese Funktion unterscheidet nicht nach Groß-/Kleinschreibung.

stripcr() Wenn im angegebenen String sich ein Newline Zeichen befindet, wird das, welches am nächsten am Anfang des Strings ist, durch ein NULL (\0) Zeichen ersetzt.

striprcr() Wenn im angegebenen String sich ein Newline Zeichen befindet, wird das, welches am nächsten am Ende des Strings ist, durch ein NULL (\0) Zeichen ersetzt.

RIPEMD Funktionen – rmd.h

Diese beiden Funktionen erleichtern den Umgang mit den RIPEMD Hashfunktionen (siehe 2.4.1 auf Seite 20).

do_ripe_md(char *file) Diese Funktion produziert einen RIPEMD-160 Hashwert der angegebenen Datei `file`. Der Wert wird auf dem Standardausgabekanal ausgegeben und an den Aufrufer zurückgegeben.

RMD_buf(char *buf,dword len) Diese Funktion produziert einen RIPEMD-160 Hashwert des übergebenen Puffers. Der Wert wird an den Aufrufer zurückgegeben.

Zugriffskontrolle – acl.h

Diese Funktionen implementieren Zugangskontrollisten (acl – access control list), die nach IP-Nummer entscheiden, ob eine Adresse “gut” oder “schlecht” ist. Diese Methode ist nicht hundertprozentig sicher, da IP-Adressen gefälscht werden können. Die Funktionen operieren mit einem Zeiger auf eine Struktur `struct acl`. Es werden nur bei einigen Funktionen Beispiele gezeigt – die restlichen werden analog dazu benutzt. Alle Funktionen (bis auf aclnew()) geben Null zurück, wenn sie erfolgreich waren, sonst einen Wert ungleich Null.

aclnew() Kreiert eine neue leere Zugangskontrolliste und gibt sie den Aufrufer zurück. Diese Funktion muß aufgerufen werden, bevor eine der anderen acl* Routinen aufgerufen wird.

```
int res;
struct acl *a;
a=aclnew*();
```

aclset(struct acl *a,struct in_addr *adr,struct in_addr *mask) Fügt eine Adresse mit Netzmaske (siehe 2.1.1) zu einer bestehenden ACL hinzu.

```
res=aclset(a,"193.141.40.0","255.255.255.0");
```

aclsetc(struct acl *a, const char *cidr) Fügt eine Adresse in CIDR-Form (siehe 2 zu einer bestehenden ACL hinzu.

```
res=aclsetc(a,"193.141.89.0/24");
```

acldelete() Löscht eine Adresse mit Netzmaske aus einer ACL.

acldeletec() Löscht einen Eintrag in CIDR-Form aus einer ACL.

aclcheck(struct acl *a, struct in_addr *adr) Überprüft, ob eine IP-Adresse in einer ACL vorhanden ist oder nicht. Gibt Null zurück, wenn die Adresse vorhanden ist.

```
res=acklcheck(a,"193.141.89.2");
if (res==0) {  /* Adresse is in der ACL */
```

aclload(struct acl *a, const char *path) Lädt eine ACL aus einer Datei.

```
res=aclload(a,"/etc/zugangs-acl");
```

acldestroy(struct acl *a) Zerstört die ACL und gibt allen Speicher zurück.

```
res=acldestroy(a);
```

Diverses – path.h, output.h

buildpath() Baut einen Pfad zu einer Datei aus den zwei übergebenen Komponenten zusammen und übergibt ihm dem Aufrufer in frisch allokiertem Speicher.

```
char *neu;
char file[]="foo.config";
#define PATH "/tmp/gaga/"
neu=buildpath(PATH,file);
```

out_open(char *who) Gibt dem Syslog bestimmte Optionen mit. `who` ist dabei eine Identifikation des Aufrufers – z.B. der Programmname.

output(int mode,char *string) Diese Funktion gibt je nachdem, wie sie aufgerufen wird, ihr zweites Argument `string` auf der Standardausgabe (`mode`!=0) oder im Syslog aus.

4.3 Directory Service, yawgmoth

yawgmoth ist die Implementierung des Directoryservers von *Mcntp*. Ein Host, der News via *Mcntp* versenden möchte, muß genau einen yawgmoth Prozeß betreiben.

Nach dem Start von yawgmoth wird eine Konfigurationsdatei eingelesen, in der einige Parameter von yawgmoth definiert sind, wie z.B. die Reichweite der Ankündigungspakete oder welche Gruppen welche TTL erhalten. Das genaue Format dieser Datei ist weiter unten im Abschnitt 4.3.1 beschrieben.

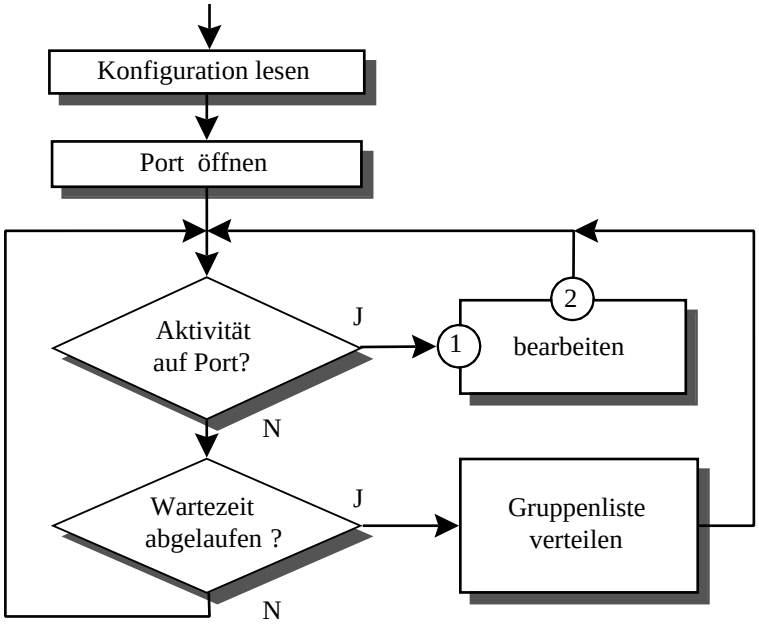


Abbildung 26: Hauptroutine von Yawgmoth

Als nächstes wird ein TCP-Port via Bibliotheksfunktion open_tcp_serv() geöffnet, der als Anlaufstelle für die Verbindungen von mcxmit dient. Als Nummer des TCP-Ports wurde von der IANA (siehe 2.6) die Nummer 5418 mit dem Namen MCNTP offiziell zugewiesen.

Zusätzlich wird ein UDP-Port geöffnet, welcher dazu dient, die Ankündigungspakete von anderen Directoryservern zu empfangen. Diese Informationen werden momentan, im Gegensatz zu Abbildung 21 im Abschnitt 3.4.2 auf Seite 29, noch nicht von yawgmoth ausgewertet (siehe auch Abschnitt 7.1 auf Seite 77). Für den Betrieb auf einem zentralen Newsserver innerhalb der Grenzen eines Providers ist dies aber nicht weiter störend.

In der Hauptschleife wird geprüft, ob sich Aktivitäten auf dem TCP-Port abspielen (Verbindungswunsch oder Verbindungsabbau) und diese dann bearbeitet.

Für Aktivitäten auf dem Port gibt es drei Möglichkeiten, wie sie in Abbildung 27 auf Seite 41 dargestellt sind. Wenn ein neuer Client eine Verbindung anfordert, wird diese geöffnet und in die Liste der offenen Verbindungen eingetragen. Wenn der Client die Verbindung beendet, sei es, daß er sich ordentlich abmeldet (siehe auch Abschnitt 4.3.2) oder daß es zum Verbindungsabbruch kommt, wird die Verbindung wieder aus der Liste entfernt.

Ansonsten wird in der Liste der offenen Verbindungen geschaut, ob auf eine dieser Verbindungen reagiert werden muß. Dies kann zum einen der beschriebenen Verbindungsabbau sein, zum anderen aber auch eine Anfrage nach einer Multicastgruppe. Diese (neue) Gruppe wird ermittelt (mehr hierzu im nächsten Abschnitt), dem Client

mitgeteilt und in der Liste vermerkt. Aus dieser Liste werden dann auch die Ankündingungspakete generiert.

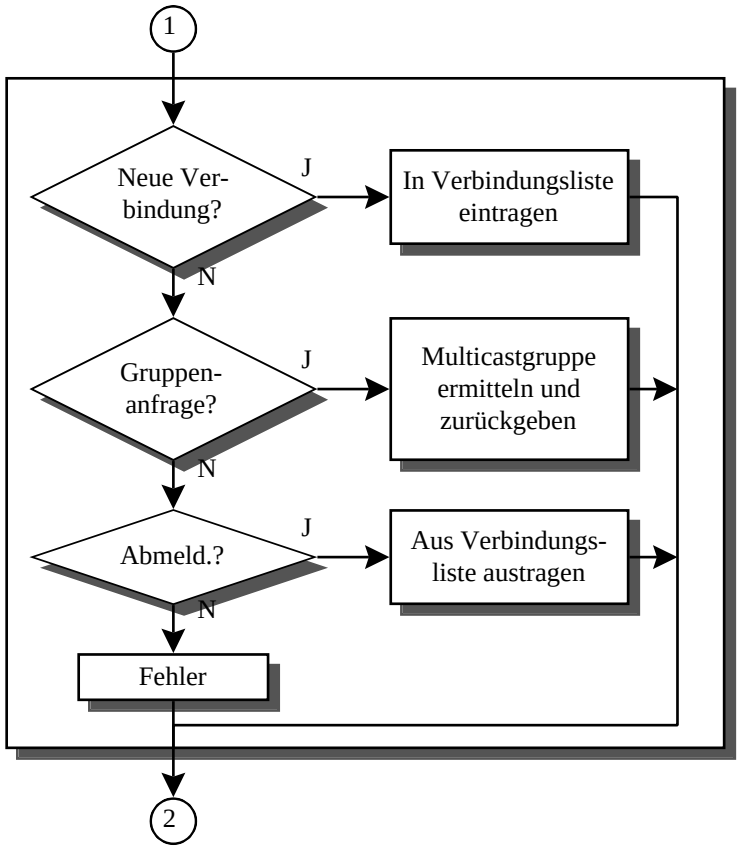


Abbildung 27: Aktivitäten auf dem TCP-Port bearbeiten

Wenn keine Aktivitäten auf dem Port zu verzeichnen sind, wird geprüft, ob die Wartezeit zwischen dem Versand der Ankündigungspakete abgelaufen ist und im Bedarfsfall die Liste der Gruppen verteilt. Die Wartezeit beträgt ca. eine Minute, um den Datenverkehr gering zu halten. Es könnte auch nach jeder Änderung in der Liste der Verbindungen ein Ankündigungspaket generiert werden, aber dies würde in der Praxis teilweise eine große Anzahl von Ankündigungspaketen in kurzer Zeit generieren[23].

4.3.1 Konfigurationsdatei von Yawgmoth

Einige Parameter von yawgmoth müssen in der Konfigurationsdatei `yawgmoth.conf` angegeben werden. Zeilen, die Leer sind, oder mit einem "#" beginnen sind dabei Kommentarzeilen und werden von yawgmoth nicht beachtet. Es gibt zwei Arten von Einträgen:

- Die "ME" Zeile, in der die TTL der Ankündigungspakete und die Sender-ID[24] angegeben wird. Der "ME" Eintrag muß genau einmal in der Konfigurationsdatei vorkommen und sieht wie folgt aus:

  ```
  ME:16:snert
  ```

[23] Wenn bei dem Newsserver INN z.B. eine neue Newsgruppe angelegt wird, werden alle Ausgabekanäle geschlossen und wieder geöffnet. Dies bedeutet aber, daß alle Multicastsender terminieren und neu gestartet werden, was dann pro Prozeß ein Ankündigungspaket bedeuten würde.

[24] Dies ist eine Identifikation des Absenders. Siehe hierzu auch die Abschnitte 4.3.3 und 4.4.1.

Die “ME” Zeile muß vor den Gruppeneinträgen in der Datei erscheinen. Gruppeneinträge, die vor der “ME” Zeile erscheinen werden ignoriert[25].

- Die Gruppeneinträge, die vorgeben, für welche Newsgruppen welche Multicastgruppen und TTL-Werte verwendet werden sollen. Ein Eintrag sieht wie folgt aus:

  ```
  Newsgruppe:MC-Start:MC-Num:TTL:
  ```

  MC-Start ist dabei die erste Multicastgruppe, die verwendet werden soll in der dotted-quad Schreibweise[26]. MC-Num ist die Anzahl der aufeinanderfolgenden Multicast Gruppen, die für verwendet werden sollen. Ein besonderer Eintrag ist die Newsgruppe “*”, die immer dann zutrifft, wenn sonst kein anderer Eintrag passend ist.

Der in der “ME” Zeile angegebene TTL-Wert ist zudem noch der Maximalwert, den die TTL-Werte der Gruppeneinträge annehmen können. Wenn ein Wert in diesen Zeilen grösser ist, so wird eine Warnung ausgegeben und auf den Wert in der “ME” Zeile gesetzt.

Diese Konfigurationsdatei `yawgmoth.conf` muß beim Start von yawgmoth vorhanden sein, andernfalls terminiert yawgmoth.

4.3.2 Kommunikation zwischen yawgmoth und mcxmit

Bei der Bearbeitung der Aktivitäten auf dem TCP-Port gibt es zwei Möglichkeiten:

- Ein Client (mcxmit), der eine Multicastgruppe erfragen möchte, teilt dem Server (yawgmoth) dies mittels des “REQ:” Kommandos mit. Der Server liefert dann, nachdem er eine Multicastgruppe ermittelt hat, diese via “USE:” beim Client ab, und trägt diese Zuordnung in seine internen Tabellen ein.
- Wenn der Client terminiert, übermittelt er dem Server mittels “QUIT:” die Anzahl der Artikel, die er bearbeitet hat, sowie die Summen der Bytes, die die Artikel ausmachen und die tatsächlich verschickt wurden. Der Server beantwortet diese Anfrage mit “OK” und gibt die entsprechenden Einträge in den internen Tabellen wieder frei.

Die Kommunikation zwischen Client und Server wird in der nächsten Abbildung nocheinmal dargestellt:

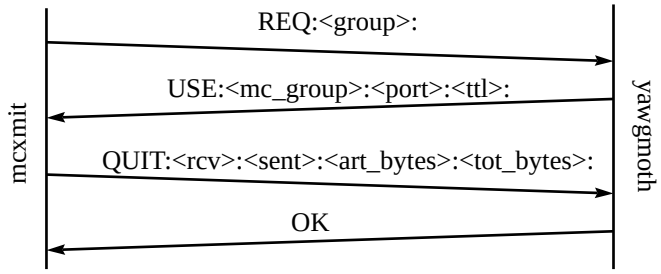


Abbildung 28: Kommunikation zwischen yawgmoth und mcxmit

Da die Kommunikation im Klartext erfolgt, ist es relativ einfach diese Kommunikation für spätere Änderungen oder Erweiterungen zu modifizieren.

[25] Es wird aber eine Warnmeldung ausgegeben.
[26] Also a.b.c.d – z.B. 228.1.2.3

4.3.3 Datenformat der Ankündigungspakete

Nachfolgend wird das Datenformat der Ankündigungspakete vorgestellt. Alle Einträge sind in Netzwerkdarstellung (siehe unten unter 4.4.1 auf Seite 45). Einträge, deren Länge nicht im Voraus bestimmt werden können, werden durch Füllbytes auf die nächste 4-Bytes Grenze aufgefüllt, um so die Verarbeitung zu erleichtern.

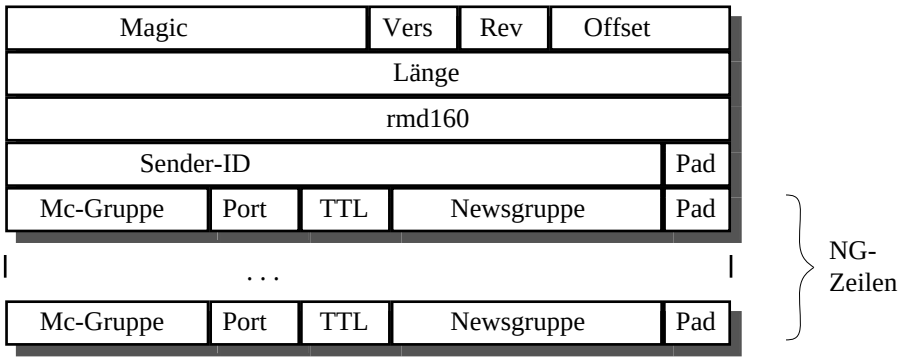


Abbildung 29: Datenformat der Ankündigungspakete

Hier folgt nun die Erklärung der einzelnen Felder:

- **Magic:** Die beiden Bytes 0xabba (16 Bit)
- **Vers:** Protokollversion (4 Bit)

  1 Für IPv4

  11 Für IPv6
- **Rev:** Protokollrevision – zur Zeit: 1 (4 Bit)
- **Offset:** Offset der NG-Zeilen (s.u.) vom Paketstart (8 Bit)
- **Länge:** Gesamtlänge des Pakets beim Versand. (32 Bit)
- **rmd160:** RIPEMD-160 Message Digest über den Rest des Paketes. (160 Bit)
- **Sender-ID:** Identifikation des Senders. Die Sender-ID ist normalerweise der Hostname des Absenderrechners mit angehängter Domain (also z.B. snert.pilhuhn.de). Dies kann aber auch der Name sein, den der Host in den *Path:* Header setzt. Bei diesem Eintrag wird nicht nach Groß-/Kleinschreibung unterschieden.
- **Pad** Füllbytes zum Auffüllen der Sender-ID auf die nächste 4-Byte Grenze
- **NG-Zeilen:** Dies sind die Zeilen, in denen die Zuordnungen von Multicastgruppe zu Newsgruppe bzw. -hierarchie festgelegt sind. Diese Zeilen werden so oft wiederholt, bis alle Gruppen angekündigt sind.
  - **Mc-Gruppe:** Die Multicastgruppe als Zahl (32 Bit bei IPv4 und 128 Bit bei IPv6)
  - **Port:** Der UDP Port, der für diese Gruppe genutzt werden soll (16 Bit).
  - **TTL:** Die Reichweite der Multicastpakete für diese Gruppe.

– **Newsgruppe:** Der Name der Newsgruppe oder -hierarchie, die verteilt wird. Dies kann auch eine Abkürzung (z.B. *cng* für die Hierarchien comp.*,news.* und gnu.*) oder sonst eindeutige Bezeichnung sein (z.B. *big8* für die als Big-8 bezeichneten Hierarchien[27]).
– **Pad:** Füllbytes zum Auffüllen einer NG-Zeile auf die nächste 4-Bytes Grenze. gestalten.

Die Füllbytes der Felder auf die nächste 4-Bytes Grenze sind notwendig, um die Bearbeitung effizienter zu gestalten bzw. überhaupt zu ermöglichen. Manche Prozessoren können keine Datenstrukturen, die größer als ein Byte sind, von einer ungeraden oder auf keine 4-Bytes Grenze ausgerichteten Adresse laden.

Die Grundstruktur des Ankündigungspaketes ist in der Headerdatei `group_select.h` in der C-Struktur *struct ypack* zu finden; die Struktur der NG-Zeilen befindet sich in *struct ngline* in derselben Datei.

4.4 Multicastsender, mcxmit

Der Multicastsender mcxmit funktioniert im Wesentlichen so, wie in Kapitel 3.7 auf Seite 31 beschrieben. Für jede Menge von Newsgruppen und -hierarchien, die in einer Multicastgruppe verschickt werden sollen, muß ein eigener mcxmit Prozess betrieben werden. Dadurch vereinfacht sich der interne Aufbau von mcxmit stark: mcxmit muß keine *Newsgroups:* Zeilen der Artikel überprüfen, um festzustellen, in welche Multicastgruppe gesendet werden soll. Diese Aufgabe erledigt bereits der NTA.
Vor Aufruf der Hauptroutine, deren grober Ablauf in der nebenstehenden Grafik dargestellt wird, wird noch der private Schlüssel zum Signieren der Artikel geladen (zum Format dieses Schlüssels siehe Abschnitt 4.6.1 auf Seite 51); ist dieser nicht vorhanden, terminiert mcxmit. Dann wird, wie schon im Abschnitt 4.3 beschrieben, vom Directory Server yawgmoth die Multicastgruppe erfragt, über die die Artikel versendet werden sollen und ein entsprechender Socket geöffnet. Dies geschieht, indem eine TCP-Verbindung zu Port 5418 auf dem Host *localhost* gemacht wird. Diese Verbindung bleibt bestehen, bis mcxmit sie beim Beenden wieder abbaut.
Die Hauptroutine liest den Pfad zu einem Artikel vom Standardeingabekanal, und stellt fest, ob der Artikel kleiner ist, als 63500 Bytes. Trifft dies zu, wird der Artikel geöffnet und in einen Puffer geladen. Wurde die Message-ID nicht über den Standardeingabekanal erhalten, wird diese aus dem Artikel extrahiert. Danach wird der Inhalt des Puffers komprimiert. Danach wird der Pufferinhalt digital signiert.

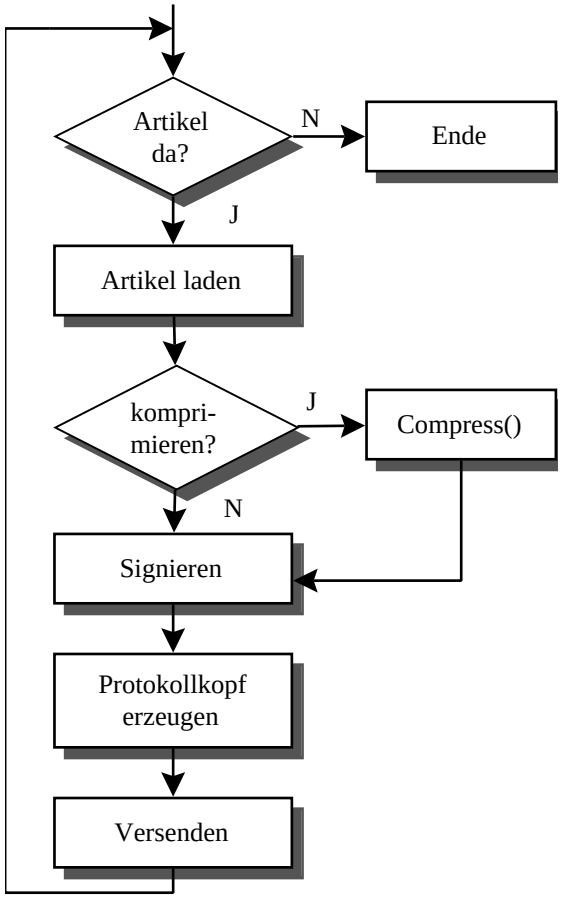


Abbildung 30: Hauptroutine von Mcxmit

[27] comp.*, humanities.*, misc.*, news.*, rec.*, sci.*, soc.*, talk.*

Der *Mcntp* Protokollkopf wird ausgefüllt und der Artikel mittels send_to() verschickt. Wenn eine (bei der Installation zu bestimmende) Menge von Artikeln verschickt wurde, gibt mcxmit eine Statusmeldung via Syslog aus. Der Aufbau eines Datenpaketes wird im nächsten Abschnitt beschrieben, ebenso, wie die Daten digital signiert werden (Abschnitt 4.4.2). Nach Verlassen dieser Hauptroutine (also wenn keine Artikel mehr auf der Standardeingabe erscheinen), meldet sich mcxmit bei yawgmoth wieder ab, gibt allokierten Speicher zurück und terminiert dann.

4.4.1 Paketformat der Artikeldaten

Nachfolgend wird das Datenformat der Pakete vorgestellt, die die Artikel transportieren sollen. Alle Einträge sind in Netzwerk Byteorder (s.u.).

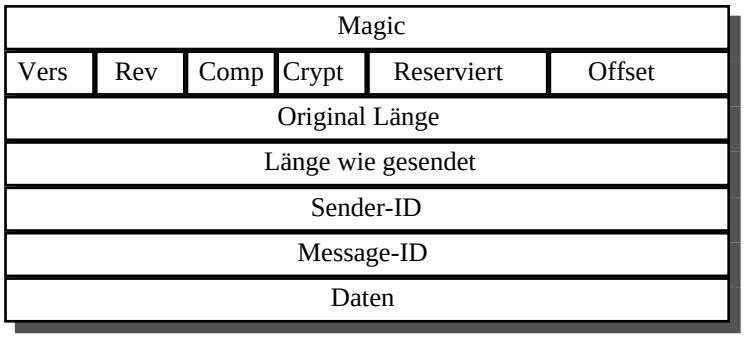


Abbildung 31: Paketformat der Artikeldaten

- **Magic:** die feste Zeichenkette “McNt” (32 Bit)
- **Ver:** Protokollversion – aktuell: 1 (4 Bit)
- **Rev:** Protokollrevision – aktuell: 1 (4 bit)
- **Comp:** Gibt an, welche Kompression benutzt wird; aktuell sind die beiden nachfolgenden Werte definiert. (4 Bit)
  - 0 Artikel ist unkomprimiert
  - 1 Artikel ist mit zlib [LD96, Deu96a, Deu96b] komprimiert.
- **Crypt:** Gibt an, ob die Artikel verschlüsselt wurden. Bisher ist noch keine Verschlüsselungsmethode festgelegt worden. (4 Bit)
- **Reserviert:** Diese Bits sind reserviert für zukünftige Protokollerweiterungen (8 Bit)
- **Offset:** Gibt an, wo im Paket die Artikeldaten (Feld “Daten” s.u.) beginnen (8 Bit)
- **Original Länge:** Originallänge des Artikels (32 Bit)
- **Länge wie gesendet:** Gesamtlänge des Pakets, wie es versendet wird. (32 Bit)
- **Sender-ID:** Eine Identifikation des Absenders. Wird genutzt, damit der Empfänger den passenden Schlüssel für die Überprüfung der digitalen Signatur benutzen kann. Die Sender-ID ist, wie bei yawgmoth, normalerweise der Hostname des Absenderrechners mit angehängter Domain (also z.B. snert.pilhuhn.de). Dies kann aber auch der Name sein, den der Host in den *Path:* Header setzt. Bei diesem Eintrag wird nicht nach Groß-/Kleinschreibung unterschieden. (NULL (\0) terminierte Zeichenkette)

- **Message-id:** Message-ID des Artikel. (NULL (\0) terminierte Zeichenkette)
- **Daten:** Die eigentlichen, digital signierten Nutzdaten nach eventueller Kompression und Verschlüsselung. Sollen die Daten komprimiert und verschlüsselt werden, ist zuerst die Kompression anzuwenden, um die Menge der Daten, die verschlüsselt werden zu verringern und die Geschwindigkeit der Artikelverarbeitung zu erhöhen. Weder mcxmit, noch mcrcv unterstützen aktuell die Verschlüsselung von Artikeln.

Die Grundstruktur des Artikelpaketes ist in der Headerdatei `mcntp_packet.h` in der C-Struktur *struct mcpack* zu finden.

Dadurch, daß die interne Darstellung von Zahlen sich auf verschiedenen Prozessorarchitekturen unterscheiden, ist es notwendig die Zahlendarstellung vor dem Versand bzw. nach dem Erhalt der Daten zu wandeln. Im Internet ist es Standard, Zahlen in der sog. *Network Byte Order* über das Netz zu transportieren. *Network Byte Order* entspricht dabei der Zahlendarstellung auf sog. *Little-Endian* Maschinen. Hierbei wird das niederwertige Byte eines Wortes oder Langwortes zuerst transportiert. Außerdem wird hierbei auch das niederwertigste Bit eines Bytes zuerst transportiert.

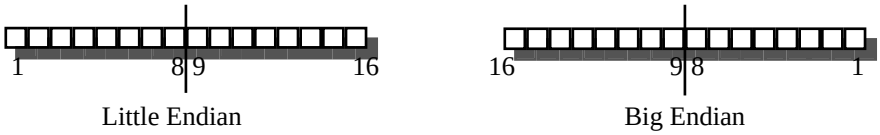


Abbildung 32: Darstellung eines 16-Bit Wortes bei Big- und Little- Endian Architekturen

Da die Reihenfolge der Bits innerhalb eines Bytes bei Big- und Little- Endian[28] unterschiedlich sind, muß hierauf in der internen Darstellung Rücksicht genommen werden, wie dies der folgend Ausschnitt aus der Struktur *struct mcpack* zeigt:

```
#ifdef BBIGEND
    char vers:4;                /* protocol version */
    char rev:4;                 /* protocol revision */
    char comp:4;                /* what compression ? */
    char crypt:4;               /* what encryption */
#endif
#ifdef BLITTLEEND
    char rev:4;                 /* protocol revision */
    char vers:4;                /* protocol version */
    char crypt:4;               /* what encryption */
    char comp:4;                /* what compression ? */
#endif
```

Abbildung 33: Ausschnitt aus der Struktur mcpack

Die Umwandlung der internen Zahlendarstellung von Worten und Langworten in die Netzwerkdarstellung erfolgt mittels der folgenden Befehle[29]:

[28] Es gibt auch noch PDP-Endian der PDP Architektur von Digital, auf denen die Ur-Unixe entwickelt wurden.
[29] Diese Befehle repräsentieren also quasi die im Internet-Schichtenmodell nicht vorhandene Darstellungsschicht des OSI-Modells.

	Worte	Langworte
von Host an Netz	htons()	htonl()
von Netz an Host	ntohs()	ntohs()

Beispiele für Little-Endian Architekturen sind Intel x86 und Digital Vax; für Big-Endian sind dies Motorola 680x0, IBM AIX-Maschinen (RS6000 und PowerPC) und Sun Sparc.

4.4.2 Signieren der Artikel

Vor dem Versand der Artikel müssen diese, wie bereits erläutert (siehe 3.7), digital signiert werden. Dies geschieht, indem ein Message-Digest von der Nachricht produziert und dieser dann verschlüsselt wird.

Da die bei *Mcntp* verwendete RSAPrivateEncrypt() Funktion einen Block von 48 Bytes auf einmal verschlüsselt, der von RIPEMD-160 stammende Message-Digest, aber nur 20 Bytes groß ist, müssen die restlichen 28 Bytes noch zusätzlich hinzugefügt werden. Dies kann geschehen, indem diese 28 Bytes zufällig erzeugt, mit einem festen Wert vorbelegt oder, wie nun vorgestellt, aus dem Artikel genommen werden.

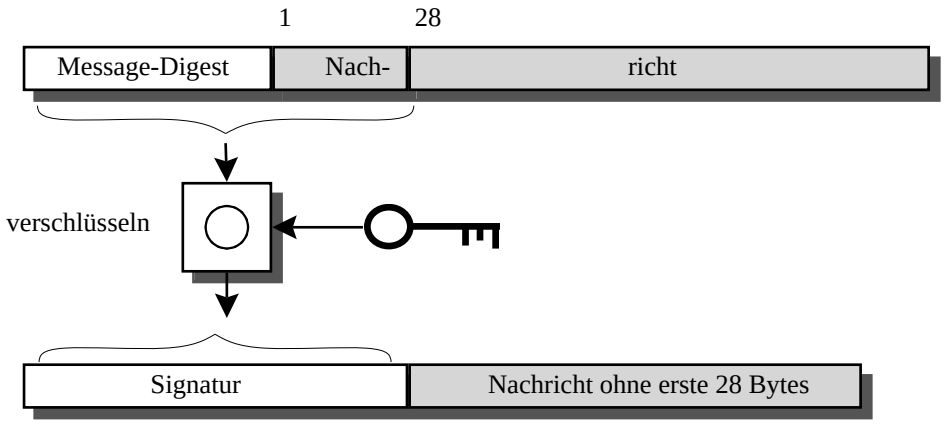


Abbildung 34: Format der digitalen Signatur

Da das Ver- und nachfolgende Entschlüsseln einer Nachricht diese unverändert lässt (also *decrypt(encrypt(n)) = n*) können zum Auffüllen des zu verschlüsselnden Puffers auch Daten aus der Nachricht genommen werden. Diese Daten werden beim Versand nicht mehr gesondert übertragen, da sie ja bereits in der Signatur enthalten sind. Der Vorteil dieses Verfahrens gegenüber der Verwendung von zufällig gewählten oder fest vorgegebenen Daten ist, daß pro Artikel 28 Bytes gespart werden, was sich bei 300.000 Artikeln am Tag zu 8 MB/Tag aufsummiert.

4.5 Multicastempfänger, mcrcv

Der Multicastempfänger mcrcv ist zuständig für den Empfang der mittels mcxmit verschickten Daten. Die erhaltenen Artikel werden dann via NNTP an einen NTA weitergeleitet, wie dies in 4.5.1 beschrieben wird.

Ist dieser NTA nicht betriebsbereit, werden die Artikel auf einer lokalen Platte abgelegt, so daß sie später, wenn der Server wieder funktionstüchtig ist, nicht noch einmal vom entfernten Newsserver übertragen werden müssen (siehe Abschnitt 4.5.2).

Nach dem Start wird die Liste der bekannten öffentlichen Schlüssel (der sog. *Keyring* zum Format siehe 4.6.1) geladen. Ist der Keyring nicht vorhanden, terminiert mcrcv. Danach wird eine Verbindung zum angegebenen NNTP-Server aufgemacht; ist keiner angegeben, wird der auf dem lokalen Host verwendet. War der Verbindungsaufbau erfolgreich, befindet sich mcrcv im Modus `M_NNTP`, ansonsten im Modus `M_Spool`. Nachfolgend wird mcrcv Mitglied in der Multicastgruppe MCNTP-DIRECTORY.MCAST.NET und wartet auf Ankündigungspakete. Wenn ein Ankündigungspaket erhalten wird, wird es durchsucht, ob es die passenden Newsgruppen enthält. Wenn nicht, wird weiter gewartet.

Wird die passende Gruppe gefunden, wird mcrcv Mitglied in der entsprechenden Multicastgruppe (“Neue Gruppe betreten” in Abbildung 35) und wartet auf Pakete. Bei Erhalt eines Datenpaketes wird als erstes die digitale Signatur überprüft. Ist diese gut, wird das Paket weiterverarbeitet, wie dies weiter unten beschrieben wird, ansonsten verworfen.

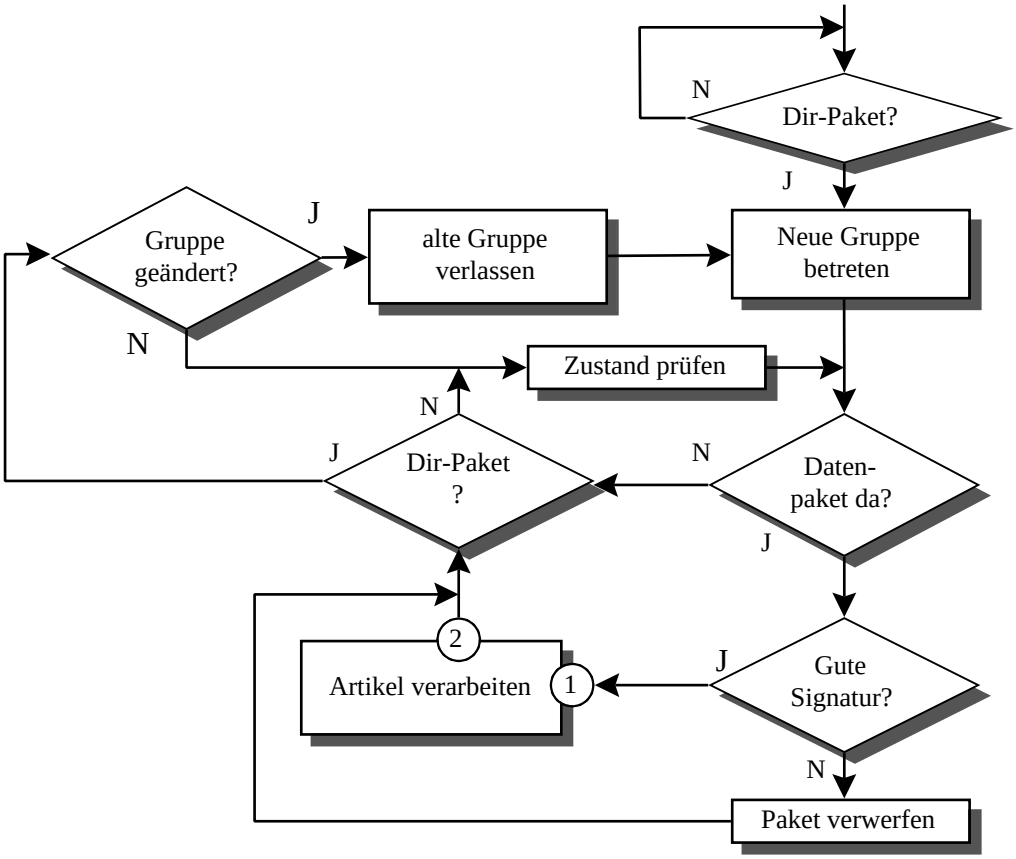


Abbildung 35: Ablaufdiagramm mcrcv

War das empfangene Paket ein Ankündigungspaket, wird geprüft, ob sich die Multicastgruppe, in der die Artikeldaten versandt werden, geändert hat. Wenn dies der Fall ist, wird die Gruppe verlassen, in der mcrcv Mitglied ist (“alte Multicastgruppe verlassen” in Abb. 35) und die im Ankündigungspaket angegebene abonniert (“Neue Gruppe betreten” in Abbildung 35).

Die Überprüfung der Signatur erfolgt, wie schon in Abschnitt 3.7 prinzipiell beschrieben, dadurch, daß der Datenbereich des Paketes (der Wert “Daten” in Abbildung 31) mit dem zur “Sender-ID” passenden Schlüssel entschlüsselt wird (der Schlüssel muß sich im *Keyring* befinden). Schlägt die Entschlüsselung fehl, wird das Paket weggeworfen. Dann wird über den empfangenen Artikel ein RIPEMD-160 Message-Digest gebildet und mit dem im Paket enthaltenen verglichen. Stimmen die beiden überein, stammt der Artikel von einem bekannten Absender und wurde beim Transport nicht verändert (Dadurch erspart man sich eine extra Prüfsumme über das Paket)..

In “Zustand prüfen” wird, wenn sich mcrcv nicht im Zustand `M_Nntp` befindet, und eine gewisse konfigurierbare Zeitspanne verstrichen ist (z.B. alle zwei Minuten), überprüft, ob der NNTP-Server wieder in der Lage ist, Artikel anzunehmen. Wenn ja, nimmt mcrcv den Betriebszustand `M_Nntp` wieder an.

Die weitere Bearbeitung des Pakets ist in Grafik 36 dargestellt. Als erstes wird das Paket, wenn nötig dekomprimiert. Dies geschieht durch Aufruf der Funktion uncompress() aus der Bibliothek zlib;schlägt dies fehl, wird das Paket weggeworfen.

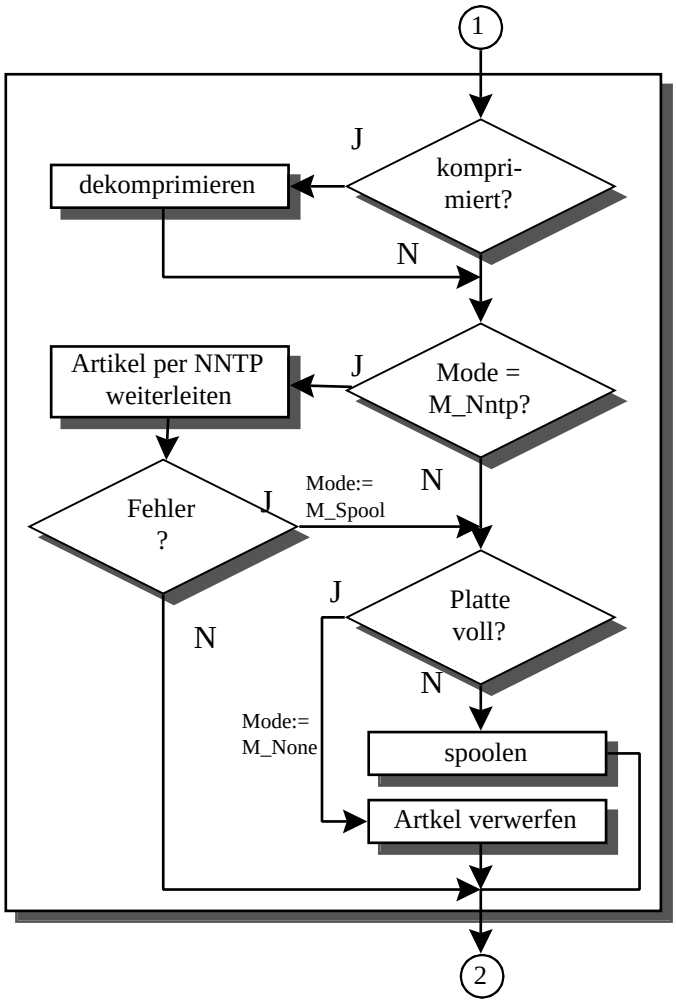


Abbildung 36: Bearbeitung der Artikel

Wenn der Server im Modus `M_Nntp` ist, d.h. der Newsserver ist in der Lage, Pakete zu empfangen, wird versucht das Paket via NNTP dem NTA abzuliefern (siehe 4.5.1. Falls dies nicht gelingt, wird, falls noch Plattenplatz vorhanden ist, der Artikel als sogenannter *Batch* auf der Platte abgelegt, wie dies unter 4.5.2 noch näher beschrieben wird und der Server geht in den Betriebszustand `M_Spool` über. Schlägt auch noch das Ablegen auf der Platte fehl, geht mcrcv in den Betriebszustand `M_None` über und verwirft das Paket.

In den Abbildungen nicht dargestellt ist ein zusätzliches Feature: bevor die Ankündigungs- und Datenpakete verarbeitet werden können diese mittels Zugangskontrollisten (ACL) nach Absender gefiltert werden. Diese Zugangskontrollisten bieten nur sehr wenig Schutz gegen die Verfälschung der Daten, da die IP-Adresse des Absenders gefälscht sein kann (deswegen werden die Artikel digital signiert). Die Listen können aber eingesetzt werden, wenn z.B. ein Störenfried Pakete schickt, um so die Last auf dem Client zu erhöhen. Mehr Informationen zu den Zugangskontrollisten gibt es in Abschnitt 4.6.3.

4.5.1 Kommunikation mit dem Newsserver

Die Kommunikation mit dem Newsserver erfolgt, wie bereits mehrfach erwähnt über NNTP (Siehe auch 2.3.2 auf Seite 16). Dabei werden die Artikel an den Server, sofern dieser Streaming-NNTP unterstützt, nicht mittels *ihave*, sondern mittels *takethis* Kommando übertragen.

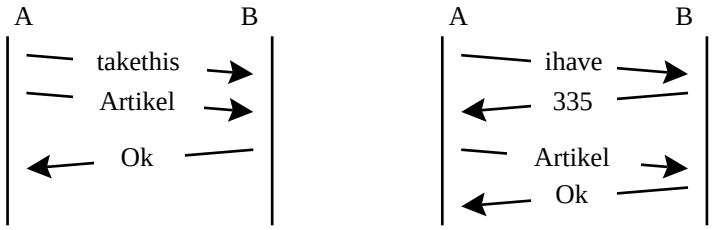


Abbildung 37: Vergleich zwischen takethis und ihave

Die Übertragung via *takethis* ist eine Optimierung, da so der NNTP-Dialog verkürzt wird und damit die Wahrscheinlichkeit steigt, daß der Artikel schon via *Mcntp* eingeliefert wurde, bevor er mittels normalem NNTP-Feed angeboten wird.

Unterstützt der Server kein Streaming-NNTP, kann mcrcv Standard-NNTP nach RFC977 [KL86] verwenden und die Artikel via *ihave* einliefern.

4.5.2 Spooling

Wenn der Newsserver nicht in der Lage ist, Artikel anzunehmen, aber noch Platz auf der Festplatte ist, kann mcrcv diese auf der Platte ablegen. Dies hat den Vorteil, daß sie später, wenn der Server wieder verfügbar ist, lokal zugestellt werden können und nicht via NNTP nachgesendet werden müssen. Diesen Prozeß wird als *Spooling* bezeichnet. Dabei werden viele Artikel in einer Datei (dem sog. *Batch*)) abgelegt und können später mittels des Programms rnews an den Newsserver weitergeleitet werden.

Damit rnews diese Dateien verarbeiten kann, müssen diese in einem bestimmten Format sein, das nun vorgestellt wird: Für jeden Artikel, der abgelegt werden soll, wird ein Header "`#! rnews n`" in eine eigene Zeile geschrieben (wobei n die Größe des Artikels angibt), gefolgt von dem Artikel selbst.

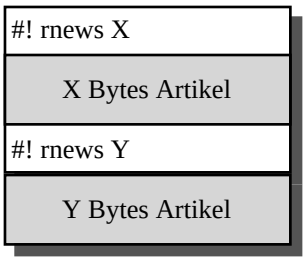


Abbildung 38: Format eines Batches

Dieses Batches werden normalerweise bei der Übertragung von Newsartikeln via UUCP verwendet. Damit sind sie bestens geeignet, die via *Mcntp* erhaltenen Artikel, die nicht via NNTP weitergeleitet werden können, temporär zwischenzuspeichern bis sie weiterverarbeitet werden können.

4.6 Sonstige Module

Die obigen drei Module (Directory Service, Sender und Empfänger) stellen den wichtigsten Teil der Arbeit dar. Für den Betrieb sind allerdings noch mehr Programme nötig, die diese drei unterstützen. Ausserdem gibt es noch Programme, die die Anwendung der Funktionenbibliothek demonstrieren.

4.6.1 Schlüsselgenerierung, keygen

Damit die Artikel digital signiert werden können, muß jeder Senderhost einen privaten und einen öffentlichen Schlüssel bereitstellen. Der private Schlüssel dient mcxmit dazu den Message-Digest der Artikel vor dem Versand zu verschlüsseln. Dieser Schlüssel darf nicht öffentlich zugänglich gemacht werden, da sonst der Sender einer Nachricht im Missbrauchsfall nicht mehr festgestellt werden kann.

Die Schlüssel werden innerhalb von keygen mittels der R_GeneratePEMKeys() Funktion der librsa, (s.u.) generiert und ein Message-Digest des Schlüssels ermittelt. Der Message-Digest kann benutzt werden, um festzustellen, ob ein Schlüssel manipuliert wurde oder beim Transport verändert wurde.
Das Programm keygen bietet noch einige Funktionen zur Bearbeitung eines Keyrings, die im Kapitel “Betrieb” noch vorgestellt werden. Diese Funktionen haben mit der Schlüsselgenerierung nichts zu tun und wurden nur der Übersichtlichkeit halber in keygen mit aufgenommen.

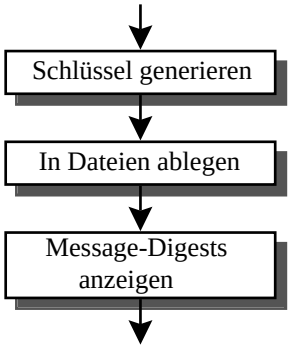


Abbildung 39: Ablaufplan Keygen

Format der Schlüssel und des Keyrings

Jeder Schlüssel besteht intern aus zwei Teilen: der Sender-ID und dem eigentlichen mittels R_GeneratePEMKeys() generierten Teil. Die Sender-ID ist eine Zeichenkette und beinhaltet, wie schon in den Abschnitten 4.3.3 und 4.4.1 beschrieben, entweder den Hostnamen des Senders in Domainschreibweise oder den Namen, den der Sender in den “Path:” Header setzt.

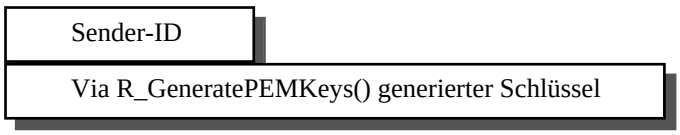


Abbildung 40: Format eines Schlüssels

Der *Keyring* ist einfach eine Datei, in der mehrere (öffentliche) Schlüssel zusammengefaßt wurden, wie dies die nächste Abbildung zeigt.

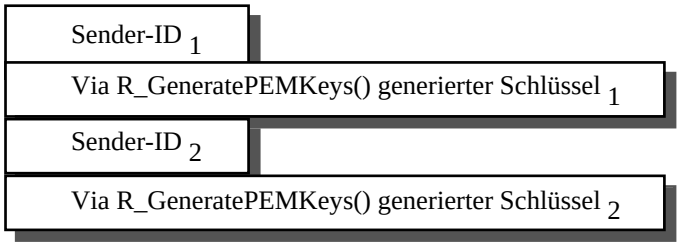


Abbildung 41: Format des Keyrings

Schlüsseltransport

Wie der öffentliche Schlüssel von einem Senderhost zu einem Empfängerhost gelangt, ist egal. Die einzige Voraussetzung, die nötig ist, ist daß das Übertragungsmedium transparent für den Transport von Daten ist, die alle acht Bits eines Bytes ausnutzen.

Eine Möglichkeit, den Schlüssel zu verteilen ist, ihn auf einem möglicherweise vorhandenen WWW-Server des Senders abzulegen, eine andere, ihn via Email oder Diskette an die Empfänger zu verteilen. Der Empfänger kann die Unverfälschtheit dann mittels des Message-Digests des Schlüssels überprüfen.

4.6.2 Ausgabe von Directory Service Paketen, groupdump

Das Programm groupdump dient als Informations- und Diagnosewerkzeug und zeigt den Inhalt erhaltener Ankündigungspakete auf dem Bildschirm an.
Nachdem groupdump Mitglied in der Multicastgruppe MCNTP-DIRECTORY.MCAST.NET geworden ist, wartet groupdump auf Ankündigungspakete, “parst” diese und gibt sie in für Menschen lesbarer Form auf dem Bildschirm aus.
Groupdump kann somit benutzt werden, um zu sehen, ob bei einem Empfänger überhaupt Ankündigungspakete ankommen. Wenn dies der Fall ist, werden die verteilten Newsgruppen zusammen mit den zugehörigen Multicastgruppen, den Empfängerports und der jeweiligen TTL-Werten angezeigt. Was groupdump nicht anzeigen kann, ist eine eventuell vorhandene Zuordnung von Abkürzungen zu den sich dahinter verbergenden Newsgruppen und -hierarchien.

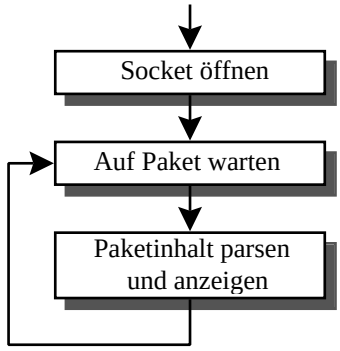


Abbildung 42: Ablaufplan Groupdump

Dadurch, daß die Multicastgruppen und Portnummern mit angegeben werden, ist es für einen Administrator leichter, nach Fehlern zu suchen, wenn z.B. Pakete für eine bestimmte Gruppe nicht beim Empfänger ankommen, weil sie auf dem Weg dorthin gefiltert wurden.

4.6.3 Überprüfung von Zugangskontrollisten, acltest

Wie bereits im Abschnitt über mcrcv erwähnt 4.5, gibt es Zugangskontrollisten (access control list, ACL), die benutzt werden können, um Pakete von bestimmten IP-Adressen zu filtern. Mit acltest steht ein Werkzeug zur Verfügung, um eine Zugangskontrolliste vor dem Einsatz auf Fehler zu überprüfen.

Zugangskontrollisten können aus einer Datei geladen werden. Die Einträge können dabei wie folgt aussehen:

Format	Beispiel
Adresse	193.141.89.1
Adresse/Länge	193.141.89.1/32
Adresse Netzmaske	193.141.89.1 255.255.255.255

Tabelle 8: Format von Zugangskontrollisten

Das Beispiel gibt dabei in allen drei Fällen dieselbe Adresse an.

4.6.4 Messung von Paketverlusten, mess

Mess besteht aus zwei Teilen: Einem Multicastsender und einem -empfänger. Damit kann gemessen werden, wie hoch der Paketverlust bei bestimmten Paketgrößen ist.

Beim Betrieb als Empfänger wird ein Multicastsocket geöffnet und auf Pakete gewartet. Bei Erhalt eines Interrupt-Signals (Control-C) bricht mess ab und gibt die Anzahl der erhaltenen Pakete und Bytes aus.

Beim Betrieb als Sender wird eine Anzahl von Paketen mit einer bestimmten Größe versandt; die Anzahl und die Größe können dabei vom Anwender bestimmt werden.

4.6.5 Generierung von RIPEMD-160 Werten, rmd

Rmd produziert RIPEMD-160 Message-Digests der auf der Kommandozeile angegebenen Dateien. Rmd ruft dabei zur Erzeugung der Digests wiederholt die Funktion do_ripe_md() auf:

```
snert!42> ./rmd rmd.c
1cf7c700ac66da851532771c02f255a9aaf6993f
```

4.6.6 Ein kleines Chatprogramm, mcchat

Mcchat ist ein einfaches Anwendungsbeispiel für die Bibliothek, libmc, das ein Chatprogramm, ähnlich talk(1) darstellt. Durch die Verwendung der Routinen aus der Bibliothek konnte es mit sehr wenigen Zeilen Code programmiert werden.

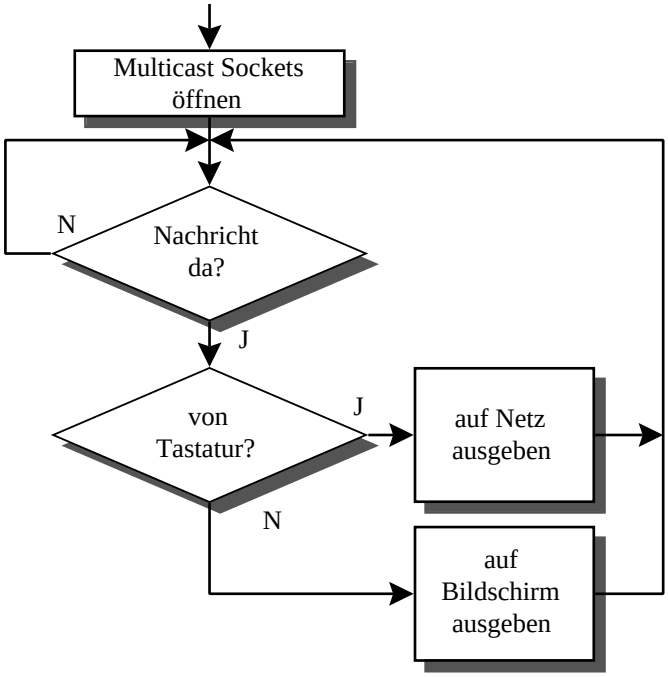


Abbildung 43: Mcchat

Als erstes werden mit den beiden Funktionen get_read_socket() und get_send_socket() die beiden Multicastsockets

für das Lesen vom Netz und das Schreiben auf das Netz geöffnet. Wie bereits in Abschnitt 2.2.2 beschrieben, kann dies nicht mit einem einzigen Socket erledigt werden.

Danach wird in einer Endlosschleife [30] mit Hilfe von select(2) geprüft, ob Daten vom Netz angekommen sind, oder der Benutzer eine Zeile eingegeben hat. Sind Daten vom Benutzer eingegeben worden, werden sie mit Hilfe von send_to() an die Multicastgruppe verschickt; kamen die Daten vom Netz, werden sie mittels recv_from() abgeholt und auf dem Bildschirm ausgedruckt.

Von den 150 Zeilen Code, die nötig waren, sind nur fünf Zeilen für die Multicastkommunikation notwendig.

Neben der Funktion als Demonstrationsobjekt für die Bibliothek, libmc, kann mcchat auch als einfacher „Leitungsprüfer“ herhalten. Damit kann auf einfache Weise festgestellt werden kann, ob überhaupt Multicastpakete von A nach B und zurück gelangen.

4.7 Zusätzliche Bibliotheken

Manche der oben vorgestellten Programme benötigen für ihre Funktion zusätzliche externe Funktionen, die normalerweise nicht in der Standard C Bibliothek vorhanden sind. Teilweise wurden diese Funktionen bereits in der Bibliothek libmc zur Verfügung gestellt. Die beiden zusätzlich benötigten Bibliotheken werden hier kurz vorgestellt:

zlib Die Zlib, die in den RFCs 1950-1952 [LD96, Deu96a, Deu96b] spezifiziert ist, enthält zwei Funktionen, compress() und uncompress(), die einen Puffer bekannter Länge komprimieren oder dekomprimieren. Diese beiden Funktionen werden in mcxmit und mcrcv benötigt, um die Artikel vor dem Versand zu komprimieren und sie nach dem Erhalt zu dekomprimieren können.

Die Zlib wurde von Jean-loup Gailly[31] und Mark Adler[32] entwickelt und kann u.a. über folgenden URL erhalten werden: http://quest.jpl.nasa.gov/zlib/

Die Zlib ist im *Mcntp* Quellbaum im Verzeichnis zlib/ enthalten.

rsaref Aus der librsaref bzw. der außerhalb frei verfügbaren librsaeuro werden drei Funktionen benötigt: RSAPrivateEncrypt(), RSAPublicDecrypt() und R_GeneratePEMKeys(). Die ersten beiden dienen dazu, einen Puffer mit den privaten Schlüssel zu verschlüsseln, bzw. mit dem öffentlichen Schlüssel zu entschlüsseln. Diese Funktionen werden in mcxmit und mcrcv benötigt. R_GeneratePEMKeys() wird in keygen benutzt, um die privaten und öffentlichen Schlüssel zu generieren.

Die rsaref Bibliothek wurde von J.S.A.Kapp[33] entwickelt und kann u.a. über folgende URL erhalten werden: http://www.repertech.com/RSAEuro.html

Weiterhin wird der Code zu RIPEMD-160 von Antoon Bosselaers[34] benötigt, der in der Distribution im Verzeichnis ripe-md160/ sich befindet und der u.a. von ftp://ftp.cert.dfn.de/ bezogen werden kann.

Dieser Code ist im Verzeichnis ripe-md160 im *Mcntp* Quellbaum enthalten und wurde mimimal modifiziert: die Funktion compress() wurde in RMDcompress() unbenannt, da ein Namenskonflikt mit der Funktion compress() der zlib aufgetreten ist.

[30] Welche durch die Eingabe von CTRL-C verlassen werden kann.
[31] gzip@prep.ai.mit.edu
[32] madler@alumni.caltech.edu
[33] rsaeuro@repertech.com
[34] antoon.bosselaers@esat.kuleuven.ac.be

4.8 Portierungen

Mcntp wurde bisher auf die folgenden Plattformen portiert:

- NetBSD 1.1 und 1.2D.
- Solaris 2.5.1.
- Linux 2.0.28, 2.0.30pre1 mit gnu libc und 2.0.30.
- BSD/OS i386 2.1 und 3.0.
- AIX 4.1 mit gcc.
- NeXTStep 3.3 auf Motorola Hardware[35].

4.9 Zusammenfassung

Dieses Kapitel beschreibt die Implementierung von *Mcntp*. Als erstes wurde die Bibliothek libmc beschrieben, die Funktionen enthält, die die anderen Module benötigen. Mit yawgmoth, mcxmit und mcrcv wurden Directory-Server, Multicastsender und -empfänger vorgestellt, sowie andere Module, entweder für *Mcntp* benötigt werden oder die als Demonstrationsprogramme dienen.

Für *Mcntp* wurden zwei Protokolle entwickelt, die zur Verteilung von Ankündigungen und zum Transport von Artikeln verwendet werden.

[35] Bisher nur die Library, jedoch nicht mcxmit oder mcrcv. Da bei NeXTStep die Socketpuffer sehr klein sind, ist die Verwendung zum Newstransport nur sehr rudimentär gegeben. Die Bibliothek *libmc* steht aber vollständig zur Verfügung. Programme, die keine großen Empfangspuffer benötigen, wie z.B. mcchat funktionieren ohne Einschränkung.

5 Installation und Betrieb

5.1 Installation

In der folgenden Installationsanleitung bezeichnet $MCNTP den Pfad im Dateisystem, in dem sich das ausgepackte *Mcntp* Paket befindet. $RSA bezeichnet analog das Verzeichnis, in dem die RSA Bibliothek sich befindet.

Installationsschritte sind:

1. Die librsa holen[36] und übersetzen Die rsaref Bibliothek wurde dem *Mcntp* Paket nicht beigelegt, da laut J.S.A. Kapp die RSAEURO Bibliothek in den USA Probleme mit Copyrightbestimmungen haben kann, während die Verwendung der RSAREF Bibliothek von RSA Data Security Inc. außerhalb der USA illegal ist. Aus dem Dokument "licence.txt" des RSAEURO Pakets:

   > 6. RSAEURO is a publication of cryptographic techniques. Applications developed with RSAEURO may be subject to export controls in some countries. If you are located in the United States and develop such applications using RSAEURO, you are advised to obtain a copy of RSAREF from RSADSI, as you may using RSAEURO infringe on Patents held by Public Key Partners of Sunnydale California.

   Die RSAEURO Bibliothek kann u.a. via anonymem ftp u.a. vom Server ftp.cert.dfn.de bezogen werden.

   Zum Auspacken des RSAEURO Archivs ist das Programm unzip nötig. Hierbei ist zu beachten, daß unzip mit der Option -a mit angegeben wird, da das Zeilenvorschubzeichen ^M, das in den Dateien vorhanden ist, viele Compiler verwirrt. Die notwendigen Schritte zum Erzeugen der Bibliothek sind:

   (a) cd $RSA/install

   (b) vi unix/Makefile

   (c) vi source/global.h

   (d) make -f unix/Makefile

   Im Verzeichnis $RSA/install befinden sich neben der Funktionenbibliothek, einige Beispielprogramme, die genutzt werden können, um festzustellen, ob die Bibliothek korrekt übersetzt wurde. Diese Beispielprogramme können mit den Scripten in $RSA/scripts/ automatisch ausgeführt werden; Referenzausgaben der Programme liegen bei, um die korrekte Funktion feststellen zu können.

2. *Mcntp* Software holen und auspacken. Die *Mcntp* Software ist via anonymem ftp vom Rechner ftp.xlink.net im Verzeichnis /pub/news/tools/ als Datei mcntp.tar.gz erhältlich.

3. Die Datei README im Verzeichnis $MCNTP lesen.

4. Die Datei `rsaref.a` aus $RSA/install/ und die Header-Dateien (*.h) aus $RSA/source/ nach $MCNTP/rsa kopieren

5. Die Datei `config.h` anlegen. Die Datei `config.h` kann durch Kopieren der Datei `config.h.dist` erzeugt werden. Die Datei `config.h` wurde nicht direkt beigelegt, um zu verhindern, daß Aktualisierungen der *Mcntp* Distribution eine vorhandene Version von `config.h` überschreiben.

[36] Siehe Abschnitt 4.7.

6. Die Datei `config.h` editieren. Die Werte in `config.h` stimmen nicht für jedes System und müssen vor dem Übersetzen von *Mcntp* angepasst werden. Hier folgt eine Beschreibung der Präprozessorsymbole in `config.h`, die angepasst werden müssen:

   BLITTLEEND/BBIGEND Dient zur Unterscheidung, ob der Prozessor des Rechners eine sog. BigEndian (z.B. Motorola 680x0 oder Sun Sparc) oder Little-Endian Maschine (z.B. Intel ix86) ist (siehe auch Abschnitt 4.4.1). Beispiel:

   ```
   #define BBIGEND
   ```

   DONTWANTZLIB Wenn keine Datenkompression via Zlib (s.u.) gewünscht wird, kann der entsprechende Code auskommentiert werden. Dies bedeutet aber auch, daß keine komprimierten Artikel akzeptiert werden können. Dieses Präprozessorsymbol sollte nur in Ausnahmefällen definiert werden. Beispiel:

   ```
   /* #define DONTWANTZLIB */
   ```

   HAVE_STATFS Wenn mcrcv Artikel auf der Platte ablegt (siehe 4.5.2), kann mittels des statfs() Betriebssystemaufrufs festgestellt werden, ob noch genug Platz auf der Festplatte vorhanden ist. Ist diesel Parameter nicht definiert, werden die Artikel solange geschrieben, bis das Dateisystem voll ist. Beispiel:

   ```
   #define HAVE_STATFS
   ```

   HAVE_MMAP Mit dem mmap() Systemaufruf können bei vielen Unix Systemen Dateien schneller in den Arbeitsspeicher geladen werden, als mittels read(). Mcxmit kann dies ausnutzen, um die Artikel zur Verarbeitung in den Speicher zu laden. Da nicht alle Systeme eine funktionierende Implementierung von mmap() haben, wird dieser Code nur bei Bedarf eingesetzt. Beispiel:

   ```
   #define HAVE_MMAP
   ```

   SPOOL_PATH Damit mcxmit die Artikel, die versendet werden sollen, finden kann, muß hier der Pfad angegeben werden, unter dem der Artikelbaum zu finden ist (gemeinhin auch als “Artikelspool” bezeichnet). Beispiel:

   ```
   #define SPOOL_PATH "/var/spool/news"
   ```

   SPOOL_PATH_IN Damit mcrcv Artikel, die nicht via NNTP weitergeleitet werden können, als Batches im Dateisystem ablegen kann, muß dieser Parameter definiert werden. Der Parameter bezeichnet das Verzeichnis, in dem Programm rnews die Batches erwartet. Bei den meisten Newssystemen ist dies SPOOL_PATH/in.coming. Beispiel:

   ```
   #define SPOOL_PATH_IN "/var/spool/news/in.coming"
   ```

   BATCH_SIZE Dieser Parameter definiert die maximale Größe der von mcrcv produzierten Batches in Bytes. Wenn der Wert auf -1 gesetzt wird, werden keine Batches produziert. Beispiel:

   ```
   #define BATCH_SIZE 50000
   ```

   SPOOL_LIMIT Dieser Wert definiert, falls der Parameter HAVE_STATFS ebenfalls definiert ist, den freien Platz im Dateisystem an, ab dessen Unterschreitung keine weiteren Artikel mehr im Dateisystem abgelegt werden. Dieser Wert muß größe sein, als der von BATCH_SIZE. Beispiel:

   ```
   #define SPOOL_LIMIT 100000
   ```

SPOOL_RECHECK Wenn mcrcv keine Artikel mehr via NNTP einliefern kann und Artikel gespoolt werden, wird nach der hier angegebenen Zeit in Sekunden nachgeschaut, ob der NNTP-Server wieder verfügbar ist. Wird dieser Wert auf -1 gesetzt, unterbleiben diese Tests. Beispiel:

```
#define SPOOL_RECHECK 2*60
```

CHECKPOINT Mcxmit und mcrcv geben nicht nur bei Terminierung, sondern auch während der Verarbeitung regelmäßig Daten über Syslog(3) aus. Dieser Parameter definiert, nach wievielen bearbeiteten Artikeln die Programme diese Daten ausgeben sollen. Für Server, die viele Artikel versenden oder empfangen ist hier ein Wert von 200 oder 500 sinnvoll. Beispiel:

```
#define CHECKPOINT 50
```

NNTP_PORT Normalerweise werden NNTP-Verbindungen zu Port 119 aufgebaut. Dieser Port wurde von der IANA für diesen Zweck offiziell zugewiesen. Dieser Port wird sowohl von Newsleseclients, als auch von anderen Transferagents für das Einliefern der Artikel genutzt. Aus Performancegründen werden hierfür auf manchen Systemen unterschiedliche Ports verwendet. Dieser Wert gibt an, auf welchem Port mcrcv die Artikel einliefern soll[37]. Beispiel:

```
#define NNTP_PORT 119
```

LIB_PATH Dieser Parameter gibt das Verzeichnis an, in dem die Konfigurationsdatei `yawgmoth.conf`, der Keyring und der private Schlüssel standardmäßig gesucht werden. Beispiel:

```
#define LIB_PATH "/usr/local/news/mcntp/"
```

PER_CLOSE Newstransferagents geben teilweise erst dann Statistiken über eingehende Verbindungen aus, wenn diese beendet worden sind. Da Mcrcv ein langlaufender Prozeß ist, wird hiermit dem NTA die Möglichkeit gegeben, diese Statistiken zu generieren. Der angegebene Wert in Sekunden beschreibt die Zeitspanne, nach der mcrcv die NNTP-Verbindung zum NTA schließt und wieder öffnet. Wenn der angegebene Wert größer als 86400 ist (ein Tag), wird er auf 86400 gesetzt. Ist der Wert kleiner als 60, wird er auf 60 gesetzt, um den NTA nicht unnötig zu belasten. Beispiel:

```
#define PER_CLOSE 60*60
```

Nach der Definition dieser Parameter werden noch einige Tests und Voreinstellungen in `config.h` durchgeführt. Hier sollte der Anwender keine Änderungen vornehmen.

7. Zlib übersetzen. Die Zlib unterliegt keinen solchen Beschränkungen, wie die librsa und ist deshalb im Verzeichnis $MCNTP/zlib vorhanden. Die Zlib zu übersetzen ist einfach:

   (a) cd zlib/zlib

   (b) ./configure

   (c) make

   (d) cd ../../

   Falls die Zlib nicht allgemein zugänglich auf dem System installiert wird, müssen evtl. die Makefiles für mcxmit und mcrcv angepasst werden, wie dies in diesen Makefiles angegeben ist.

8. *Mcntp* übersetzen. Dies geschieht entweder durch Aufruf von make für Systeme, die ein 4.4BSD kompatibles make haben. Ansonsten ist make mit der Option -f Makefile.<Systemtyp> aufzurufen.

[37] Dies muß bei einer Zweiteilung der Port, auf dem die Artikel von anderen Servern ankommen und nicht der, den die Newsleseclients benutzen.

9. Die Programme müssen in den Standardsuchpfad des Benutzers “news” kopiert werden, damit sie gefunden werden, ohne daß zum Aufruf jedesmal der komplette Pfad mit angegeben werden muß.

10. Das Verzeichnis LIB_PATH, das in der Datei `config.h` definiert wurde, muß bei Bedarf noch angelegt werden.

5.1.1 Portierungen auf andere Plattformen

Mcntp wurde bereits auf die in Abschnitt 4.8 beschriebenen Systeme portiert. Wenn *Mcntp* auf andere Systeme portiert werden soll und nicht die BSD-Version von make, pmake zum Einsatz kommt[38], muß im Hauptverzeichnis ein neues Makefile für dieses System mit dem Namen *Makefile.*<*System*> angelegt werden. Dieses Makefile erhält man am besten, indem man eines der bestehenden Makefiles kopiert. In diesem Makefile sind Änderungen an folgenden Eintragen zu machen:

CC Der Pfad zum verwendeten C-Compiler. Beispiel:

```
CC=gcc
```

AR Der Pfad zum verwendeten ar Programm. Mit ar werden einzelne Dateien zu einer Bibliothek zusammengefügt. Beispiel:

```
AR=/usr/ccs/bin/ar
```

RANLIB Der Pfad zum verwendeten ranlib Programm. ranlib kreiert eine Tabelle der externen Referenzen einer Bibliothek und wird nicht auf allen Systemen benötigt. Beispiel

```
RANLIB=/bin/ranlib
```

ELIBS Bibliotheken, die zusätzlich eingebunden werden müssen. Beispiel:

```
ELIBS="-lfoo -lbar -L/lib/such/pfad -lbaz"
```

RSAREF Die Linker Anweisungen, die nötig sind, um die rsaref Bibliothek in mcrcv und mcxmit einzubinden. Wenn die Eurorsa Bibliothek wie oben beschrieben installiert wurde, ist dies:

```
RSAREF="-L../rsa/ -lrsaref"
```

Wenn *Mcntp* auf ein neues System portiert wurde, sollten die dazu nötigen Änderungen an

<mcntp-bugs@pilhuhn.de>

geschickt werden, damit sie in spätere Versionen von *Mcntp* einfließen können.

[38] pmake bietet neben der Möglichkeit, daß das Übersetzen der Quellen parallel erfolgen kann, noch eine weitere Besonderheit. pmake kann Steuerdateien und andere Makefiles via einer `.include` Direktive einbinden. Die diese Dateien enthalten dann die Regeln, die pmake benutzt, um Programme zu übersetzen. Diese Regeln gelten systemweit. Dadurch sind die Makefiles meist sehr klein.

5.2 Betrieb

Nach dem die *Mcntp* Software übersetzt und installiert ist, kann *Mcntp* in Betrieb gehen. Die folgenden Abschnitte zeigen, wie die einzelnen Module in der Praxis eingesetzt werden. Der Betrieb von *Mcntp* setzt eine vorhandene multicastfähige Infrastruktur voraus. Wie eine solche Infrastruktur bereitgestellt wird, ist nicht Teil dieser Arbeit[39].

Wenngleich auf einem Host *Mcntp* sowohl als Sender, als auch als Empfänger eingesetzt werden kann, ist diese Konstellation eher unüblich. Im Folgenden wird deswegen zuerst der Betrieb als Sender und dann der als Empfänger begesprochen. Die Beispiele kommen alle vom Betrieb bei Xlink.

5.2.1 Betrieb als Sender

Um Artikel via *Mcntp* versenden zu können, muß auf dem Senderhost neben den Multicastsendern mcxmit auch ein Directoryserver yawgmoth betrieben werden. Außerdem muß ein Schlüsselpaar erzeugt werden, welches zur Generierung und Überprüfung der digitalen Signaturen benötigt wird.

Directory Server starten

Der Directory-Server yawgmoth wird beim Systemstart gestartet. Dies kann z.B. aus einem der Scripten heraus geschehen, die das System beim Start automatisch ausführt (`/etc/rc`). Da yawgmoth nicht als Benutzer “Root” gestartet werden muß, ist `rc.news` ein geeigneter Platz.

```
> yawgmoth &
```

Zu Debuggingzwecken kann mit der Option -d eine ausführlichere Ausgabe auf dem Bildschirm ausgegeben werden.

```
> yawgmoth -d 1
Sat Aug 23 22:35:41 MET DST 1997
REQ:pilhuhn.test
USE:228.1.1.1:50001:8:
QUIT:1:0:0:0:
OK
```

So sieht man am Beispiel, welche Anfragen nach Multicastgruppen von den Sendern erfolgen (REQ), welche Gruppe und Port der Client erhält (USE) und welche Abschlußmeldungen der Client liefert (QUIT). Siehe auch Abschnitt 4.3.2 auf Seite 42.

Für die Funktion von yawgmoth ist die Konfigurationsdatei `yawgmoth.conf` notwendig, welche standardmäßig im Verzeichnis LIB_PATH gesucht wird (siehe auch 4.3.1). Eine minimale Version sieht wie folgt aus:

```
ME:8:blackbush
*:228.1.1.1:250:16:
```

[39] Ein multicastfähiger Rechner, auf dem *Mcntp* überstetzt wird, ist bereits eine solche (minimale) Infrastruktur und kann für Tests der Software bereits genutzt werden.

In der “ME” Zeile wird die Sender-ID auf *blackbush* und die TTL auf 8 gesetzt. Der einzige vorhandene Gruppeneintrag trifft auf alle Gruppen zu und instruiert, yawgmoth die Multicastgruppen 228.1.1.1 bis 228.1.1.250 zu verwenden. Da der TTL-Eintrag größer ist, als der in der “ME” Zeile, wird er auf den dortigen Wert gesetzt.

Sollen Newshierarchien mit unterschiedlichen geographischen Ausdehnungen verteilt werden, kann dies über die TTL Werte gesteuert werden. Im folgenden Beispiel wird noch die für Karlsruhe lokale Hierarchie *ka.** verteilt, wobei die Reichweite der Verteilung beschränkt ist (TTL von vier):

```
ME:16:blackbush
# Die Ka.*  Gruppen sollen in Karlsruhe bleiben
ka.*:239.1.1.1:4:
# Die anderen Gruppen im gesamten Backbone verteilen
*:228.1.1.1:250:16:
```

Die Zeilen, die mit dem “#” Zeichen beginnen, sind Kommentarzeilen; ihr Inhalt wird von yawgmoth nicht beachtet.

Auswahl der passenden Multicastgruppenbereiche

In Abschnitt 2.2.4 auf Seite 11 wurde die Unterteilung der Multicastadressen in global und lokal gültige Adressen beschrieben. Wie in diesem Abschnitt und in 2.2.3 erläutert, ist die Angabe der TTL nötig, aber nicht ausreichend, um die Ausbreitung zu beschränken und die Gruppen lokal zu halten.

Newsgruppen, die nicht im globalen MBone verteilt werden, sollten deshalb im Bereich der administrativ limitierten Gruppen verteilt werden, um die Ausbreitung dieser Gruppen zu limitieren. Dies gilt nicht nur für Gruppen, die für ein geographisches Gebiet lokal sind (z.B. für eine Stadt), sondern auch wenn die Gruppen nur für Kunden eines Internetserviceproviders (ISP) verfügbar sein sollen.

Im letzten Beispiel wird die *ka.** Hierarchie bereits in einer der administrativ begrenzten Multicastgruppen verteilt, während alle anderen Gruppen im globalen MBone sichtbar sind[40].

Key generieren

Damit mcxmit funktionieren kann, muß vor dem ersten Versenden ein Schlüsselpaar generiert werden. Dieses Paar besteht aus einem privaten Schlüssel zur Bildung der digitalen Signaturen *mcntp-key.priv* und dem öffentlichen Schlüssel *mcntp-key.pub*, der bei den Empfängern zur Überprüfung der Signaturen verwendet wird.

```
> keygen -b 512 -i snert
Generating keys with 512 Bit length .. may take a while
Ripe-md160 fingerprint for the public key is :
71d897cff15e4edb7a3f10e592a4579739aa2e5a
> ls -ls m*
1 -rw-r--r--  1 hwr  pilhuhn  712 Aug 23 19:48 mcntp-key.priv
1 -rw-r--r--  1 hwr  pilhuhn  264 Aug 23 19:48 mcntp-key.pub
```

[40] Durch die TTL von 16 allerdings maximal 16 Hops vom Sender entfernt.

Bei der Schlüsselgenerierung muß neben der Sender-ID noch die Länge des zu generierenden Schlüssels mit angegeben werden. Je länger der Schlüssel ist, desto sicherer ist er gegenüber Versuchen, ihn kryptographisch zu brechen. Allerdings bedeutet eine größere Schlüssellänge auch ein deutlich höher Aufwand [41] bei der Bildung und Verifikation der digitalen Signatur:

Länge in Bit	Signieren in Sekunden	Überprüfen in Sekunden
512	0.35	0.04
768	1.11	0.07
1024	2.62	0.13

Tabelle 9: Einfluß von Schlüssellängen auf die Geschwindigkeit des Signierens

Da die digitale Signatur bei *Mcntp* nur benutzt wird, um zu verhindern, daß auf einfache Weise Massenwerbepostings oder gefälschte Artikel in Umlauf kommen, sind in der Praxis Schlüssellängen von 512 Bit ausreichend. Im Usenet gibt es viele Server, die für jedermann zugängliche sind. Die Benutzung dieser Server ist mit weniger Aufwand versehen, als das kryptographische Brechen des Schlüssels.

Der generierte private Schlüssel muß vor Benutzung durch mcxmit nach LIB_PATH/mcntp-key.priv kopiert werden.

Ein Senderhost kann mehr als einen Schlüssel haben. So wäre es beispielsweise möglich, eine private Newsgruppe mit einer anderen Sender-ID zu versenden, als die allgemein zugänglichen Gruppen. Wenn nun der öffentliche Schlüssel dieser privaten Gruppe nur an berechtigte Empfänger verteilt wird, werden nur diese in der Lage sein, diese Artikel via mcrcv zu empfangen. Da die Artikel selbst nicht verschlüsselt werden, ist diese Methode aber nicht vollkommen sicher – die Daten können immer noch durch Programme, die auf der Netzwerkschicht Pakete "mithören", empfangen werden[42].

Sender starten

Um den Sender mcxmit zu starten, muß diesem der Name der Gruppe als Parameter mitgegeben werden. Der Sender wartet dann auf der Standardeingabe auf Daten vom NTA.

```
> mcxmit -g pilhuhn.test
```

Die Eingabe besteht aus dem Pfad in Dateisystem, unter dem der Artikel im Dateisystem gefunden werden kann[43], und der Message-ID. Die Angabe der Message-ID ist nicht unbedingt nötig, erspart mcxmit den Aufwand, diese herauszufinden, während diese von den NTAs ohne Aufwand mitgeliefert werden.

Wenn die Artikel vor dem Versand komprimiert werden sollen, ist die Option -c anzugeben.

```
> mcxmit -g pilhuhn.test -c
```

[41] Die Zeiten in Tabelle 9 wurden auf einem 486dx2/80 PC ermittelt.

[42] Wenn die versendeten Daten vor dem Versand komprimiert werden, wird bei der Erzeugung der digitalen Signatur ein Teil des Kompressionsbaums, den zlib benutzt, ebenfalls verschlüsselt, so daß es schwieriger wird, die Nachricht zu lesen, wenn der Schlüssel zum Entschlüsseln der Signatur nicht bekannt ist. Dies ist allerdings kein kryptographisch sicheres Verfahren zur Datenverschlüsselung! Vergleiche auch Abschnitte 4.4.2 und 2.4.1.

[43] Dies kann ein relativer Pfad sein. in diesem Fall wird dem Pfad der Wert von SPOOL_PATH vorangestellt.

Im Betrieb ist es aus Performancegründen[44] besser, mcxmit direkt vom NTA aus mit den Artikelpfaden zu versorgen. Hierzu ist es nötig, daß der NTA selbst Prozesse starten kann, die ihre Informationen dann über die Standardeingabe erhalten. Beim NTA INN ist dies über einen sogenannten *Channel-Feed* möglich. Ein Eintrag in der Datei `Newsfeeds`[45] wird dann wie folgt aussehen:

```
de-cast:de.*:Tc,Wnm,<30000:mcxmit -g de -c
```

Die Felder dieses Eintrags, die voneinander durch einen Doppelpunkt ":" getrennt sind, bestehen aus einem internen Namen des Empfängers (de-cast), der Liste der Gruppen, die der Empfänger erhalten möchte (de.*), einer Liste von Optionen und einem Parameter, der von den Optionen im dritten Feld abhängt. In diesem Fall bedeutet "Tc," daß dieser Eintrag ein *Channel-Feed* ist und der Parameter im vierten Feld ein Programm ist, welches auf dem Standardeingabekanal auf Daten wartet. Die Option "Wnm" gibt an, daß ein Artikelpfad relativ zum Newsspool und die Message-ID ausgegeben werden soll; "<3000" gibt an, daß nur Artikel weitergeleitet werden sollen, die kleiner als 30.000 Bytes groß sind. Mcxmit und mcrcv sind in der Lage, bis zu 60 kBytes große Artikel zu verarbeiten, jedoch ist es in der Praxis sinnvoll, den Betrieb auf Artikel bis maximal 30.000 Bytes zu beschränken, da die Empfangspuffer der Empfängersysteme oftmals nicht groß genug sind, mehrere große Artikel zu halten und dadurch erhaltenen Pakete verworfen werden. Diese Beschränkung verringert die Anzahl der transportierten Artikel nur geringfügig (siehe auch Abschnitt 2.3.1 auf Seite 14).

Dieser Feed verteilt nur Artikel in der Hierarchie *de.**, die vor dem Versand komprimiert werden. Damit INN das mcxmit Programm finden kann, muß es im Standardsuchpfad zu finden sein. Ist dies nicht der Fall, ist beim Aufruf der gesamte Pfad zu mcxmit anzugeben.

Mcxmit gibt über Syslog(3) zur Laufzeit Daten aus, wieviele Artikel bisher vom NTA erhalten und via mcxmit verschickt wurden. Außerdem enthalten diese Daten Angaben, wie groß die Artikel vor dem Versand waren und wieviele Bytes verschickt wurden.

```
Aug 24 18:17:46 blackbush mcxmit[6708]: mcxmit: 149 received 149
 sent 386356 art_bytes 258115
```

In diesem Beispiel wurden 149 Artikel vom Server angeboten. Alle 149 Artikel konnten im Dateisystem gefunden werden[46] und waren kleiner als 60kBytes. Die Artikel hatten vor dem Versand zusammen 386356 Bytes; 258115 Bytes wurden über das Netz verschickt (Artikeldaten und *Mcntp* Protokollheader gesamt).

5.2.2 Betrieb als Empfänger

Für dem Empfang von NetNews via *Mcntp* wird neben dem Multicastempfänger mcrcv noch der öffentliche Schlüssel des Senders benötigt. Dieser Schlüssel kann z.B. vom Sender auf einem World-Wide-Web Server abgelegt worden sein, wie dies bei Xlink der Fall ist.

[44] Wenn die Artikelpfade in eine Datei geschrieben und erst später durch mcxmit verarbeitet werden, ist die Wahrscheinlichkeit höher, daß die Artikel bereits auf anderem Wege beim Empfänger angekommen sind und damit die Übertragung vergeblich war.

[45] Dies ist beim NTA INN die Datei, in der festgelegt wird, welcher Empfänger welche Gruppen erhalten soll.

[46] Dies muß nicht notwendigerweise der Fall sein, da der Artikel vor dem Weiterversand bereits mittels *Cancel* Kontrollnachricht (siehe 2.3) gelöscht worden sein kann.

Keyring erzeugen

Der Keyring wird von *Mcntp* standardmäßig in LIB_PATH gesucht; keygen nutzt diesen Pfad nicht automatisch, um zu verhindern, daß bei Tests der private Schlüssel oder der Keyring überschrieben wird. Falls noch kein Keyring vorhanden ist, muß ein leerer Keyring angelegt werden:

```
> touch keyring
```

Der neue Schlüssel kann dann dem Keyring mittels des Kommandos keygen -r *keyring* -a *Schlüssel* hinzugefügt werden.

```
> keygen -a ./snert-key.pub -r ring
> keygen -a ./blackbush-key.pub -r ring
```

Sollte bereits ein Schlüssel mit gleicher Sender-ID vorhanden sein, meldet keygen dies und bricht den Vorgang ab:

```
> keygen -a ./snert-key.pub -r ring
Warning: key for <snert> already exists. Not replacing it. (use -f )
```

Mit der Option -l kann man mit keygen feststellen, welche Schlüssel auf einen Keyring vorhanden sind:

```
> keygen -l -r ring
id        fingerprint
-------------------------------------
snert     f3c967d39347abe76497ab3b1d625d6b255fe295
blackbush eac6614400992e4c0306fb80417b7979cc3984d8
```

Für jeden Schlüssel werden neben der Sender-ID ein RIPEMD-160 Message-Digest des Schlüssels mit ausgegeben. Durch Vergleich dieses Fingerabdrucks mit dem beim Sender kann festgestellt werden, ob die Schlüssel korrekt übertragen wurden.

Welche Gruppen gibt es?

Normalerweise erhält man vom Sender eine Liste der Gruppen, die dieser verteilt. Ist dies nicht der Fall, kann diese Liste mit groupdump erhalten werden.

```
> groupdump
------ Sat Aug 23 16:37:30 1997
 >>193.141.89.2<<
Packet Version: 1.1 Sender-id: >snert<
pilhuhn.test:228.1.1.1:50001:1
------ Sat Aug 23 16:37:36 1997
 >>193.141.40.8<<
Packet Version: 1.1 Sender-id: >blackbush<
rec:228.1.1.50:50050:16
de:228.1.1.52:50052:16
hss:228.1.1.53:50053:16
```

In diesem Beispiel gibt es zwei Sender, *Snert* und *Blackbush*. Snert verteilt nur eine Gruppe “pilhuhn.test”, während Blackbush mehrere verteilt. Während “de” und “rec” die Namen von Newshierarchien sind, ist “hss” eine Abkürzung für “humanities, soc, sci”. Diese Zuordnung wurde willkürlich gewählt.

Empfänger starten

Um den Multicastempfänger mcrcv zu starten, muß mit der Option -g dem Empfänger die zu erhaltende Newsgruppe mitgegeben werden.

```
> mcrcv -g de &
```

Der Empfänger testet, ob er Artikel auf dem NNTP-Server des Rechners “localhost”[47] einliefern kann. Ist dies der Fall, werden die Artikel dort eingeliefert, ansonsten auf Platte abgelegt. Mit der Option -s kann ein anderer Server und mit der Option -p ein anderer Port, als der in NNTP_PORT voreingestellte, gewählt werden:

```
> mcrcv -g rec -s news.xlink.net -p 1234&
```

Der Keyring wird von mcrcv standardmäßig in LIB_PATH/keyring erwartet. Mit der Option -K kann auch ein anderer Keyring verwendet werden:

```
> mcrcv -g comp -K /pfad/zu/einem/keyring &
```

Normalerweise gibt mcrcv seine Ausgaben via Syslog(3) aus. Mit der Option -d wird die Ausgabe auf den Bildschirm ausgegeben; je größer der mitgegebene Parameter ist, desto detaillierter ist die Ausgabe. Mit -d 5 wird z.B. für jeden erhaltenen Artikel das Ankunftsdatum, der Sender, die Message-ID und Größe, sowie der Antwortcode des NNTP-Servers angegeben:

```
> mcrcv -d 5 -g de
Aug 23 19:29:59. blackbush ==> <5tn3qa$lbe$5@gwdu19.gwdg.de>  239 1130
Aug 23 19:30:00. blackbush ==> <5tn450$lbe$6@gwdu19.gwdg.de>  239 1352
Aug 23 19:30:01. blackbush ==> <5tn60j$lbe$7@gwdu19.gwdg.de>  239 2248
^Cmcrcv: 3 offered 3 accepted 0 rejected 0 refused 0 spooled
```

Im Normalfall wird beim Systemstart eine Reihe von mcrcv Prozessen gestartet, die die gewünschten Newsgruppen bearbeiten. Wie bereits bei yawgmoth ist `rc.news` ein geeigneter Platz, die Empfänger zu starten. Diese laufen dann bis zum Herunterfahren des Systems.

Mcrcv gibt ebenfalls Betriebsdaten über Syslog(3) aus. Diese Daten geben an, wieviele Artikel mcrcv erhalten hat und wie diese weiterverarbeitet wurden.

```
Aug 24 18:24:01 snert mcrcv[21498]: check: 21950 offered 21291
    accepted 659 rejected 0 refused 0 spooled
```

Hier wurden 21950 Artikel vom Netz empfangen. Davon wurden 21291 vom Newsserver akzeptiert, während 659 vom Server als bereits vorhanden zurückgewiesen wurden. Es wurde kein Artikel vom Server nachträglich abgewiesen und auch kein Artikel auf Platte abgelegt, weil der Server nicht verfügbar war.

[47] localhost ist bei Unix Systemen der Name der sog. Loopbackadresse, 127.0.0.1.

5.3 Möglicher Betrieb im globalen MBone

Der Betrieb von *Mcntp* innerhalb des abgeschlossenen Bereichs eines einzelnen Internetserviceproviders (ISP) unterscheidet sich von Betrieb im globelen MBone:

- Innerhalb eines ISP kann der Bereich der administrativ limitierten Adressen verwendet werden (siehe 2.2.4) über den ein ISP volle Kontrolle hat.
- Der ISP kann eine oder mehrere Maschinen als Multicastsender bestimmen und hat eine Kontrolle darüber, welche Gruppen verteilt werden.
- Das Monopol eines einzelnen Providers als Sender im MBone wird von der Internetgemeinde nicht anerkannt werden. Wenn mehrere Provider als Sender auftreten, besteht die Gefahr, daß Artikel mehrfach übertragen werden und damit Bandbreite verschwendet wird.
- Multicastgruppen im MBone werden dynamisch verteilt. Es gibt, wie in 2.2.4 beschrieben, eine Reihe von Multicastadressen, die von der IANA (siehe auch Abschnitt 2.6) zugeteilt werden. Bis auf diese werden die Gruppen von Programmen, die IP-Multicast nutzen, dynamisch und nach einem Pseudozufallsprinzip vergeben. Damit kann nicht ausgeschlossen werden, daß zwei unterschiedliche Applikationen dieselben Multicastgruppen nutzen.

Um NetNews auf dem MBone zu verteilen, bietet sich folgendes Vorgehen an:

- Es werden entweder bei IANA eine Reihe von Multicastadressen für diesen Zweck reserviert oder in Zusammenarbeit mit den Entwicklern anderer Software versucht, einen Bereich für den Newstransport freizuhalten.
- Die großen Backboneserver, die News via *Mcntp* versenden wollen, dürfen nur die Artikel via *Mcntp* weiterleiten, die von ihnen selbst oder von Teilnehmern stammen, die ihre Artikel nur über diesen Sender verteilen können. Artikel, die über andere Backbonenewsserver erhalten werden, dürfen nicht via *Mcntp* weiterverteilt werden.
- Die Artikel werden mit einer TTL von 128 weltweit verteilt.
- Die Zustellung der Artikel via NNTP wird beibehalten.

Durch diese Vorgehensweise werden keine Artikel doppelt über *Mcntp* verteilt, da jeder Artikel nur von dem Backboneserver eingespeisst wird, der dafür zuständig ist. Durch die Beschränkung der Sender auf die Backboneserver, ist die Menge der Schlüssel, die bei den Empfängern bekannt sein muß noch relativ überschaubar.

Damit das aufgezeigte Szenario funktioniert, muß allerdings yawgmoth noch dahingehend modifizert werden, daß yawgmoth für den Versand von Artikeln Gruppen nutzt, die bereits von anderen Sites annonciert werden.

5.4 Zusammenfassung

Dieser Kapitel beschreibt, wie *Mcntp* installiert wird, welche Anpassungen an das lokale System nötig sind und wie *Mcntp* auf neue Systeme portiert werden kann. Danach wird gezeigt, wie die einzelnen Komponenten von *Mcntp* für den Newstransport bei einem Internetserviceprovider eingesetzt werden können. Zuletzt wurde ein Szenario beschrieben, um *Mcntp* auf dem globalen MBone einzusetzen.

6 Ergebnisse

Um *Mcntp* beurteilen zu können, empfiehlt es sich, *Mcntp* mit den Vorgaben aus Kapitel 3 zu vergleichen:

- Plattformunabhängigkeit
- 8-Bit-Clean
- Sparsamkeit bezüglich Bandbreite
- Ausfalltoleranz
- Transportsicherheit
- Benötigte Rechenzeit

6.1 Messungen und Tests

Alle Messungen wurden auf einem Teil des Backbones des Internetproviders Xlink in Karlsruhe in Zusammenarbeit mit den Partnerfirmen vor Ort (POPs) durchgeführt. Die untere Karte zeigt diesen Teil mit den entsprechenden Serverrechnern und Routern. Bei diesen Tests kam *Mcntp* auf unterschiedlichen Plattformen zum Einsatz.

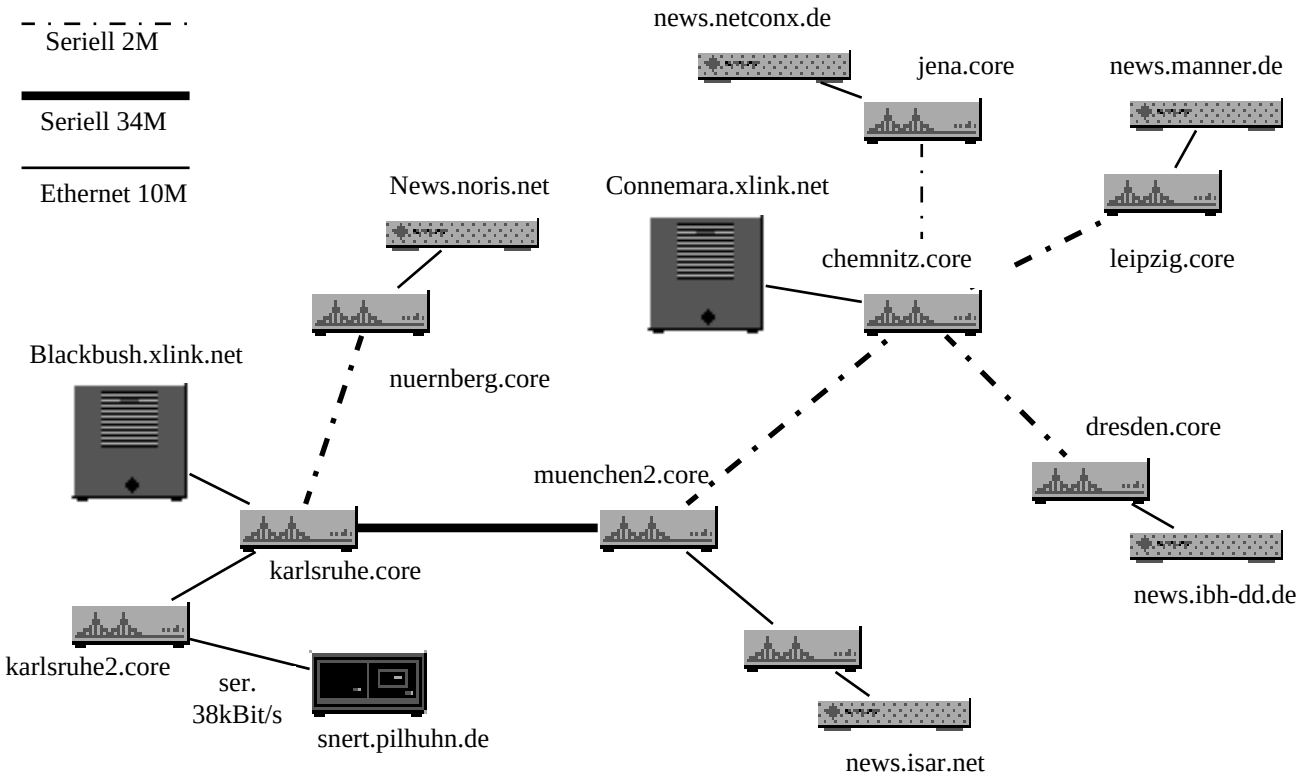


Abbildung 44: Versuchsbackbone

6.2 Plattformunabhängigkeit

Mcntp wurde auf die in Abschnitt 4.8 erwähnten Betriebssysteme bereits portiert. Da keine Funktionen verwendet werden, die für diese Betriebssysteme speziell sind, ist zu erwarten, daß auch Portierungen auf andere Betriebssysteme problemlos sind.

Mcxmit erwartet seine Eingabe über den Standardeingabekanal. Wenn nun der NTA nicht in der Lage ist, diesen selbst zu betreiben (*Channel-Feed* siehe 5.2.1), kann der NTA eine Datei mit den Pfadangaben und Message-IDs schreiben, die mcxmit dann über Umleitung der Standardeingabe zur Bearbeitung erhält. Dieser Mechanismus ist nicht auf das Unix Betriebssystem beschränkt, sondern funktioniert beispielsweise auch unter Windows NT[48] oder VMS.

Da mcrcv dem Empfänger NTA die Artikel via NNTP übermittelt, muß mcrcv nicht auf dem gleichen Rechner laufen, wie der Newsserver. Die Möglichkeit, Batches (siehe 4.5.2) zu schreiben, bietet die Möglichkeit, den Empfänger selbst auf Newssystemen zu betreiben, die keine NNTP-Unterstützung bieten[49].

Die einzige Bedingung für den Betrieb vom *Mcntp* , die erfüllt sein muß ist, daß IP Multicast auf den Rechnern unterstützt wird, auf denen mcxmit, mcrcv und yawgmoth laufen sollen. Diese Bedingung ist aber bei allen modernen Betriebssystemen erfüllt.

6.3 8-Bit clean

Mcntp ist 8-Bit clean. Dies bedeutet, daß keine Zeichen beim Transport verändert werden. Die verschickten Artikel kommen also beim Empfänger an, wie sie auf dem Senderhost verschickt wurden. Da bereits die Funktionenbibliothek *libmc* 8-Bit clean ist, war es auf Applikationsebene nicht nötig, manche Zeichen für den Transport umzukodieren.

6.4 Bandbreitenersparnis

Für den Betrieb im Backbone wurden mehrere *mcxmit* Prozesse generiert, die auf dem Newsserver *Blackbush-.xlink.net* die Artikel verteilen (siehe auch Abschnitt5.2.1). Zusätzlich wurden zwischen *Blackbush.xlink.net* und *Connemara.xlink.net*, sowie zwischen *Connemara.xlink.net* und den *news.ibh-dd.de* verzögerte NNTP-Feeds eingerichtet, um eventuell verlorengegangene Artikel nachzuliefern. Die verteilten Newsgruppen waren: Big-8[50] ohne comp.binaries.* und de.*.

6.4.1 Einfluß von Paketverlusten

Während mittels *Mcntp* nur Artikel verteilt wurden, die kleiner als 30.000 Bytes groß sind, wurden über den NNTP-Feed auch Artikel transportiert, die größer sind. Sowohl zwischen den Servern *Blackbush* und *Connemara*, als auch zwischen *Connemara* und *news.ibh-dd.de*, zeigte sich, daß die Anzahl der Artikel, die via NNTP transportiert wurden, im Bereich von 1% der Gesamtmenge lagen. Dies zeigt, daß die Annahme, daß nur mit wenig Paketverlust zu rechnen ist, sich bestätigt.

[48] Für Windows NT kann es sein, daß dazu die GNU-Win32 Werkzeuge von http://www.cygnus.com/misc/gnu-win32/ benötigt werden.
[49] C News beispielsweise wird ohne NNTP-Unterstützung verteilt.
[50] comp.*, news.*,rec.*,sci.*,soc.*,humanities.*,misc.*,talk.*

Um herauszufinden, wie groß der Paketverlust zwischen *Blackbush.xlink.net* und *Connemara.xlink.net* in der Praxis ist, wurde der verzögerte NNTP Feed von Karlsruhe nach Chemnitz auf exakt die gleiche Artikelmenge eingestellt, wobei der NNTP-Feed mit fünf Sekunden Verzögerung betrieben wurde. Hierbei zeigte sich folgendes deutliches Bild:

Artikelgröße	Artikel verschickt	Artikel via *Mcntp* erhalten	Artikel nachgeliefert	Paketverlust
<30.000 Bytes	12.000	11.992	8	0.6 Promille
<60.000 Bytes	28.000	27.936	64	2.2 Promille

Tabelle 10: Artikelverluste in der Praxis bei unterschiedlichen Artikelgrößen

Es zeigt sich also, daß im praktischen Gebrauch der Artikelverlust unter einem Prozent[51] und damit die Voraussetzung gegeben ist, *Mcntp* effizient einzusetzen. Der etwas höhere Verlust bei größeren Artikeln läßt sich damit erklären, daß zum einen hier die Wahrscheinlichkeit höher ist, daß einzelne Fragmente des Pakets verloren gehen, und zum anderen damit, daß die Netzwerkpuffer im Kern des Zielsystems nicht groß genug waren, um mehrere große Pakete, die nacheinander ankamen, zu puffern.

Nebenbei: da *Mcntp* die herkömmliche Verteilung über NNTP nicht ersetzt, sondern ergänzt, können Artikel, die via *Mcntp* nicht zugestellt wurden, über NNTP nachgeliefert werden. Es gehen also insgesamt keine Artikel verloren.

6.4.2 Datenkompression

Wie in Abschnitt 4.4 beschrieben, ist mcxmit in der Lage, die Artikel vor dem Versand zu komprimieren. Die Auswertung der von mcxmit generierten Statistikdaten führt dabei zu folgenden Bild:

		Ohne Kompression	Mit Kompression
Anzahl versendete Artikel		33044	1037905
Artikeldaten in Bytes	Bytes	68105941	2260938548
Verschickt total	Bytes	71372945	1349773580
Differenz	Bytes	-3267004	911164968
Durchschn. Artikelgröße	Bytes	2061	2178
Durchschn. verschickt (pro Art.)	Bytes	2159	1300
Kompressionsfaktor		1	1.7

Tabelle 11: Einfluß von Datenkompression auf die Artikelgrößen

Durch die Kompression der Artikel vor dem Versand kann also das übertragene Volumen auf ca. 60 % des Originalvolumens reduziert werden – wie sich dies auf die zur Übertragung notwendige Datenrate auswirkt, zeigt die nächste Tabelle :

Die Anzahl der Artikel entspricht dabei mit 100.000 der Anzahl derer, die aktuell[52] am Tag via *Mcntp* verteilt werden.

Mit diesen Daten lässt sich nun berechnen, wie die Einsparung auf der Leitung zwischen München und Chemnitz ist, wenn die Newsserver *news.manner.de* in Leipzig, *news.ibh-dd.de* in Dresden und *news.netconx.de* in Jena statt

[51] Natürlich unter der Voraussetzung, daß auf den Leitungen selbst keine Paketverluste auftreten
[52] Stand 22.7.1997

		Ohne Kompression	Mit Kompression
Artikelgröße	Bytes	2000	2000
Artikel pro Tag		100.000	100.000
Artikelvolumen ohne *Mcntp*-Header	MBytes	190,7	190,7
Artikelvolumen mit *Mcntp*-Header	MBytes	200,1	200,1
Volumen verschickt	MBytes	200,1	124,6
Datenrate	kBit/s	19	12

Tabelle 12: Einfluß von Datenkompression auf die Datenrate

via NNTP aus Karlsruhe , via *Mcntp* versorgt werden und nur die Artikel, die verloren gegangen sind, via NNTP nachgeliefert werden (natürlich können diese beiden Server auch direkt vom Server *Connemara.xlink.net* versorgt werden, was diese Einsparung auf der Leitung von München nach Chemnitz auch bringen würde, aber dieser neue Server steht erst seit kurzem zur Verfügung. Außerdem lässt sich das Rechenmodell auch auf andere Bereiche im Backbone anwenden, wo kein Server wie *Connemara.xlink.net* zur Verfügung steht (vgl. auch 3.8 auf Seite 31).

Wenn die Artikel nur mittels NNTP von Karlsruhe nach Dresden, Leipzig und Jena gelangen, beträgt die Datenmenge 200 MB pro Tag und Feed also 600 MB am Tag (bzw. 56,9 kBit/s) auf der Leitung von München nach Chemnitz.

Unter der Annahme, daß nur 95 % aller Artikel via *Mcntp* transportiert werden und die restlichen 5 % der Artikel ca. 15 % des Volumens ausmachen [53], ergibt sich die Datenmenge, die transportiert wird, wie folgt (Tabelle 13):

	Volumen	Bandbreite
Artikel via *Mcntp*	106 MB	10,1 kBit/s
Rest via NNTP (3*)	90 MB	8,5 kBit/s
Gesamt	196 MB	18,6 kBit/s
Einsparung	404 MB	38.3 kBit/s

Tabelle 13: Rechenbeispiel: tatsächliche Einsparung

Wie man sieht, ist beim Newstransport via NNTP eine 64kBit Leitung schon nahezu ausgelastet. Wenn die Artikel via *Mcntp* ausgeliefert werden, und nur fehlende oder (für *Mcntp*) zu große Artikel via NNTP nachgeliefert werden, ist die Leitung nicht einmal zur Hälfte belastet.

6.4.3 Einfluß von Verzögerungen bei NNTP

Als nächstes wurde die Verzögerung der Auslieferung zwischen *Blackbush.xlink.net* und *Connemara.xlink.net*, *Connemara.xlink.net* und *Micky.ibh-dd.de = news.ibh-dd.de*, sowie *Blackbush.xlink.net* und *Micky.ibh-dd.de* variiert, wie dies in Tabelle 14 auf Seite 73 dargestellt ist.

Bei dieser Messung wurden den jeweiligen Empfängern von den jeweiligen Sendern parallel zur Übertragung via *Mcntp* die selben Artikel via NNTP zugestellt. Diese Zustellung erfolgte mittels nntplink, wobei die Übertragung via NNTP um verschiedene Zeitspannen verzögert wurde. Bei einer Verzögerung von kleiner gleich einer Sekunde ist der Transport via NNTP zwischen *Blackbush* und *Connemara* deutlich schneller, als der Transport via *Mcntp*. Dies hängt vorallem damit zusammen, daß *Mcntp* viel Zeit auf das Komprimieren der Artikel und vorallem auf

[53] Diese Zahlen wurden unter der Berücksichtigung der Tatsache gemacht, daß die meisten Artikel, die großër als 30 oder gar 60 kBytes sind, in alt.* zu finden sind. Außerdem wurde Tabelle 5 auf Seite 16 berücksichtigt.

		Blackbush an Connemara		Connemara an Micky	
Verzöge-rung	Via NNTP gesendet	via NNTP akzeptiert	Quote	Via NNTP akzeptiert	Quote
0s	1000	960	96%	234	23,4 %
1s	1000	967	96,7%	72	7,2 %
2s	1000	336	33,6%	49	4,9 %
3s	1000	158	15,8%	35	3,5 %
4s	1000	53	5,3%	10	1,0 %
5s	1000	2	0,2%	1	0,1 %

Tabelle 14: Effekt von verzögerten NNTP-Verbindungen

die Erzeugung und Verifikation der digitalen Signatur verwendet. Ab fünf Sekunden Verzögerung sind dann alle Artikel (bis auf wenige, die auf dem Transportweg verlorengegangen sind) via *Mcntp* geliefert worden.

Die Anzahl der Artikel, die *Connemara* bei *Micky* einliefern kann, selbst wenn der NNTP-Transfer ohne Verzögerung betrieben wird, ist deutlich kleiner, als zwischen *Blackbush* und *Connemara*. Dies hat damit zu tun, daß die via *Mcntp* verteilten Artikel auf beiden Hosts dekodiert werden müssen. Erst danach kann *Connemara* den Artikel via NNTP weiterleiten. In dieser Zeit ist der Artikel bereits auf *Micky* angekommen und wird auch dort dekodiert.

Bei beiden Messungen fällt auf, daß erst bei einer Verzögerung von ca. fünf Sekunden fast alle Artikel via *Mcntp* übertragen werden. Es scheint, als ob die Router auf dem Weg die Artikel teilweise recht lange zwischenpuffern. Dies ist allerdings schwer meßbar, da die Router selbst keine Vorrichtungen dafür haben.

6.4.4 Einfluß von Crosspostings

Crosspostings sind Artikel, bei denen mehr als eine Gruppe im *Newsgroups:* Header eingetragen ist. Damit erscheint der Artikel dem Leser, als ob er in beide Gruppen gepostet wurde, wird aber nur einmal übertragen und auch nur einmal im Dateisystem abgelegt.

Da bei *Mcntp* verschiedene Newsgruppen getrennt übertragen werden können, kann es passieren, daß ein Artikel, der ein Crossposting in zwei Gruppen ist, damit auch zweifach übertragen wird (z.B. Ein Artikel ist ein Crossposting nach *de.rec.motorrad* und *rec.motorcycles.harley*. Es werden *de.** und *rec.** getrennt verteilt, wie in Abschnitt 5.2.2 gezeigt.). Damit wird die doppelte Bandbreite zum Transport benötigt[54].

Dem NNTP-Server angeboten	489154 Artikel	100%
Vom NNTP-Server akzeptiert	461213 Artikel	94,28%
Vom NNTP-Server zurückgewiesen	27908 Artikel	5,72%

Tabelle 15: Duplikate aufgrund von Crosspostings

Die Werte aus Tabelle 15 stammen vom Host *Connemara* und wurden aus den Statusmeldungen generiert, die mcrcv produziert. Es wurden also 5% der Artikel wegen Crosspostings doppelt von Karlsruhe nach Chemnitz übertragen. Der Bandbreitenbedarf der Artikel ist also im Gegensatz zu den Daten in Tabelle 13 noch 5% höher und beträgt 10,6 kBit/s anstelle von 10,1 kBit/s. Dies ist gegenüber dem Wert von 56,9 kBit/s, die ein reiner NNTP-Feed aus Karlsruhe benötigen würde, immer noch sehr gut.

[54] Da beide Artikelkopien die gleiche Message-ID haben, wird nur einer von beiden vom Empfängernewssystem akzeptiert. Somit entstehen keine Duplikate beim Empfänger.

Selbst dann, wenn die Empfänger in Leipzig, Dresden und Jena ausschließlich von Chemnitz aus versorgt würden und nur der Newssserver in Chemnitz Artikel via NNTP aus Karlsruhe erhalten würde, wäre der Bandbreitenbedarf mit 16,8 kBit/s auf der Strecke Karlsruhe Chemnitz immernoch deutlich höher. Dies kommt daher, daß NNTP im Gegensatz zu *Mcntp* die Artikel vor dem Versand nicht komprimieren kann.

Der Versand der Artikel via *Mcntp* ist also, obwohl manche Artikel mehrfach übertragen werden, wesentlich effektiver als die Übertragung via NNTP.

6.5 Ausfalltoleranz

Der Newstransport ausschließlich über *Mcntp* ist nicht ausfallsicher, da kein gesichertes Multicastprotokoll verwendet wird. Wenn aber, wie es in dieser Arbeit beschrieben wurde, ein NNTP-Feed später den Empfängern die Artikel, die sie möglicherweise erhalten möchten, nochmals anbietet, ist der Newstransport sehr ausfallsicher. Der limitierende Faktor ist dabei der NNTP-Server auf dem Sendersystem. Fällt dieser aus, können weder Artikel via *Mcntp*, noch via NNTP verschickt werden.

Die zum Transport der Newsartikel via *Mcntp* verwendeten Programme (mcxmit, mcrcv und yawgmoth) haben sich im Praxisbetrieb als stabil laufend gezeigt. Es treten keine Abstürze auf und die Programme yawgmoth und mcrcv laufen vom Start bis zum Herunterfahren des Systems ohne zutun durch. mcxmit wird mehrmals täglich vom NTA neu gestartet[55], deswegen sind hier keine so großen Laufzeiten am Stück zu erwarten.

Der Newstransport via *Mcntp* läuft im Backbone von Xlink nun seit mehr als einem Monat stabil ohne Störungen oder Ausfälle.

6.6 Transportsicherheit

Durch die Verwendung von digitalen Signaturen ist *Mcntp* gegen eine unbefugte Einspeisung von Artikeln gewappnet. Artikel werden nur dann von einem Empfänger akzeptiert, wenn sie von einem bekannten Sender signiert wurden. Der Aufwand, den zur Bildung der Signatur benutzen RSA-Schlüssel cryptograhisch zu brechen ist relativ hoch. Vorallem in Hinblick darauf, daß es weltweit viele Newsserver gibt, die offen für alle zugänglich sind.

Die digitale Signatur bietet nur die Gewißheit, daß ein Artikel von einem bekannten Sender kommt und während des Transports nicht verändert wurde. Die Signatur bietet aber keinen Schutz gegen unbefugtes Mitlesen von privaten Artikeln, wie sie z.B. bei der Verteilung von privaten Newsgruppen via *Mcntp* auftreten.

Die Verschlüsselung der Artikel ist noch nicht in mcxmit und mcrcv implementiert. Es ist aber bereits ein Feld im Protokollkopf vorgesehen (siehe Abbildung 31 in Abschnitt 4.4.1 auf Seite 45), so daß es nicht sehr viel Aufwand ist, dies zu mcxmit und mcrcv hinzuzufügen.

6.7 Benötigte Rechenzeit

Die von *Mcntp* benötigte Rechenzeit hängt in erster Linie von der Rechenleistung des verwendeten Prozessors und der Qualität der verwendeten RSA-Bibliothek ab. Eine Version, bei der die interne Arithmetik in Assembler kodiert ist, läuft schneller ab, als eine Version, bei der die Routinen in der Programmiersprache C programmiert wurden.

[55] Dies ist eine Folge der inneren Abläufe von INN.

Das Programm redemo, welches der rsaeuro Bibliothek beiliegt, benötigt beispielsweise 10% mehr Rechenzeit, um eine Datei zu verschlüsseln, wenn es mit der C und nicht mit der Assembler Version übersetzt wurde[56].

mcxmit, welches die digitale Signatur erzeugt, muß hierzu den RIPEMD Fingerprint verschlüsseln (siehe auch 2.4.1 und 4.4.2). Dies ist aufwendiger, als das Entschlüsseln, das in mcrcv zur Verifikation der Signatur nötig ist.

mcxmit benötigt auf *Blackbush* (Sun Ultra Sparc System) bei vier Artikeln pro Sekunde (dies entspricht 345.000 Artikeln am Tag) ca. 30% CPU. Die RSA Bibliothek ist hierbei eine C Version, da in der Dokumentation zur Bibliothek steht, daß die Sparc Assembler Version nicht auf allen CPUs korrekt läuft. Es wurden im Testbetrieb schon bis zu acht Artikel pro Sekunde versandt.

Auf *Connemara*, ebenfalls eine Sun Ultra Sparc, benötigen die Empfängerprozesse zusammen maximal 10% CPU – typischerweise weniger als 2%.

Die mcxmit Prozesse benötigen relativ viel CPU Leistung – vorallem auch relativ zu den mcrcv Prozessen. Da aber die Newstransferagents im Normalfall genug CPU-Leistung zur Verfügung haben[57], ist dies nicht weiter störend. Falls die CPU-Leistung nicht ausreicht, alle Artikel mit nur einer CPU zu verarbeiten, können die Artikel auch von mehreren Maschinen aus verteilt werden.

Mcntp ist auf jeden Fall leistungsfähig genug für das aktuelle Newsvolumen und hat noch Luft für zukünftige Entwicklungen.

Ein interessanter Effekt im Zusammenhang mit der Benutzung von *Mcntp* auf Senderservern liegt darin, daß die nötige Eingabe-/Ausgabeleistung zu den Festplatten hin sinkt. Dies geschieht, obwohl *Mcntp* zusätzlich zu den vorhandenen NNTP-Verbindungen betrieben wird. Beim Transport via NNTP wird derselbe Artikel nicht gleichzeitig an alle Empfänger verteilt, sondern diese Zeitpunkte variieren. Damit muß der Artikel mehrfach von der Festplatte in den Arbeitsspeicher geladen werden. Beim Versand via *Mcntp* geschieht dies nur einmal (bzw. wenn der Artikel sich noch im Plattenpuffer befindet, garnicht – siehe auch Abschnitt 2.3.4). Nachfolgende NNTP-Transfers, die benutzt werden, um verlorengegangene Artikel nachzuliefern, müssen nur noch für die Artikel, die tatsächlich übertragen werden, auf die Festplatte zugreifen.

6.8 Probleme

Eine kleine Unschönheit gibt es leider auch im Zusammenhang mit *Mcntp*: Artikel, die auf einem Host produziert werden, der die entsprechende Newsgruppe via *Mcntp* empfängt, werden wieder zu diesem Host zurücktransportiert. Beispiel: Es wird ein Artikel auf *Connemara* erzeugt. Dieser Artikel wird von *Connemara* an *Blackbush* weitergeleitet. *Blackbush* verteilt den Artikel via *Mcntp* und dieser wird wieder zu *Connemara* transportiert.

Die einzige Möglichkeit dies zu verhindern, die aber nicht gut skaliert und deshalb in der Praxis nicht einsetzbar ist, wäre, alle Artikel direkt auf dem Server zu posten, der diese später via *Mcntp* verteilt. Allerdings verhindert dies nur, daß ein Host einen Artikel, der auf ihm gepostet wurde, diesen nochmals via *Mcntp* erhält. Dieses Vorgehen verhindert nicht, daß Netzwerkbandbreite mehrfach benötigt wird, weil zum Posten auf dem Senderhost eine NNTP-Verbindung dorthin gemacht werden muß, um den Artikel einzuliefern. Der Unterschied zu einer Transportverbindung via NNTP besteht nur in den Kommandos die dafür genutzt werden.

Dieses Problem ist allerdings in der Praxis nicht so groß, wie es hier aussieht. Die in Tabelle 15 gezeigten Duplikate sind alle Duplikate - es kann nicht festgestellt werden, ob diese durch Crosspostings erzeugt werden, die

[56] auf i486dx2/80

[57] Die Newstransferagents benötigen vorallem sehr viel Bandbreite für die Festplattenzugriffe. CPU Leistung wird von den NTAs nur wenig gefordert.

mehrfach verschickt wurden oder dadurch, daß Artikel, die *Connemara* an *Blackbush* geschickt hat, wieder via *Mcntp* bei *Connemara* angekommen sind.

6.9 Zusammenfassung

Dieses Kapitel beschreibt das Verhalten von *Mcntp* im praktischen Einsatz. *Mcntp* erfüllt die in Kapitel 3 geforderten Eigenschaften und läuft stabil im Praxisbetrieb bei dem Internetserviceprovider Xlink.

7 Ausblick und Danksagungen

7.1 Mögliche weitere Entwicklungen

In diesem Abschnitt sollen noch einige mögliche weitere Entwicklungen vorgestellt werden.

Reliable Multicast Falls in Zukunft ein funktionierendes zuverlässiges Multicastprotokoll entwickelt wird, sollte *Mcntp* umgeschrieben werden, damit dieses genutzt werden kann. Im Prinzip beschränkt sich das Umschreiben dabei auf die Funktionen der Bibliothek libmc, die mit der Multicastkommunikation zu tun haben. Durch Verwendung von Reliable Multicast kann dann auch das Nachsenden via NNTP entfallen.

Verbesserung der Gruppenauswahl Programme, wie SDR (Session Directory) dienen im MBone dazu, Übertragungen von Ereignissen anzukündigen und die entsprechenden Multicastkanäle zu belegen. yawgmoth könnte diese Ankündigungen in seiner eigenen Gruppenauswahl berücksichtigen, um Kollisionen mit bereits belegten Gruppen zu vermeiden. Dies kann jedoch nicht komplett gegen Überschneidungen schützen, sondern die Wahrscheinlichkeit dazu nur verringern. Im Gegenzug sollte yawgmoth die für den Newstransport gewählten Gruppen dann auch über das Protokoll von SDR verteilen.

Backup Funktionalität Das normale Verhalten von yawgmoth ist es, Gruppen, die schon belegt sind, zu meiden. D.h. für jede neue Newsgruppe wird auch eine neue Multicastgruppe aus dem angegebenen Bereich genommen. Teilweise könnte auch ein anderes Verhalten richtig sein, um z.B. auf diese Weise den in 5.3 beschriebenen MBone Betrieb zu realisieren. Dazu müßte eine weitere Konfigurationsoption für yawgmoth in `yawgmoth.conf` hinzugefügt werden:

use Falls für die angegebene Newsgruppe bereits eine Multicastgruppe existiert, soll diese benutzt werden (siehe 5.3). Wird diese Newsgruppe noch nicht verteilt, wird eine neue Multicastadresse dafür verwendet.

new Für jede Anfrage eine eigene, neue Multicastgruppe verwenden. Dies ist das aktuelle Verhalten von yawgmoth.

stop Wenn eine Anfrage kommt, mcxmit mitteilen, daß in dieser (News-)Gruppe nicht gesendet werden darf. Dazu muß mcxmit ebenfalls modifiziert werden.

standby Wird die angegebene Gruppe bereits verteilt, wird mcxmit mitgeteilt, daß nicht gesendet werden soll. yawgmoth beobachtet weiter die Ankündigungen. Wird die angegebene Gruppe nicht mehr verteilt, gibt yawgmoth ein Zeichen an mcxmit, daß nun gesendet werden soll. Dazu muß mcxmit ebenfalls modifiziert werden. Dieser Betriebsmodus könnte genutzt werden, um auf diese Weise ein automatisches Backup zu realisieren.

Ein Beispiel für eine modifizierte Konfigurationsdatei `yawgmoth.conf` könnte wie folgt aussehen:

```
ME:8:blackbush
# de.* in vorhandenen Gruppen verteilen
de.*:228.1.1.1:250:16:use
# comp.* nur dann verteilen, wenn es niemand anders tut
comp.*:228.1.1.1:250:16:standby
# fuer die ka.* Gruppen immer eine neue MC-Gruppe nehmen
ka.*:239.1.1.0:16:4:new
# Alle anderen Gruppen nicht versenden
*:228.1.1.1:250:16:stop
```

Cancel Nachrichten verteilen Im *Usenet* wird versucht, möglichst viele der Massenwerbepostings zu löschen. Dies geschieht mittels der *Cancel* Kontrollnachricht (siehe auch Abschnitt 2.3). Wenn die *Cancel* Kontrollnachricht auf einem System eintrifft, nachdem der Artikel bereits eingetroffen ist, wird er Artikel aus dem lokalen Spool entfernt. Trifft die Cancel Nachricht jedoch schon vorher ein, wird der Artikel, der gelöscht werden soll, nicht erst vom Zielsystem akzeptiert. Es ist also gut, wenn die *Cancel* Nachrichten schneller verteilt werden, als die zugehörigen Artikel. Im Protokollkopf (siehe Abbildung 31 in Abschnitt 4.4.1 auf Seite 45) sind acht Bit bisher nicht genutzt und als "Reserviert" gekennzeichnet. Ein Bit könnte als "Cancel" Bit definiert werden. Ist dieses Bit gesetzt, ändert sich das Paketformat wie folgt:

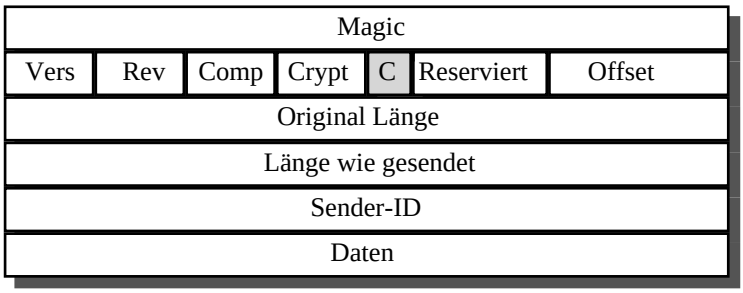


Abbildung 45: Paketformat für Cancel Nachrichten

Der Unterschied zum normalen Artikelformat, wie in Abbildung 31 definiert, liegt darin, daß die Message-ID fehlt. Dafür enthält der Datenteil eine Reihe von Message-IDs, die durch Leerzeichen getrennt sind und die, wie in Abschnitt 4.4.2 beschrieben, digital signiert sind.

Ein solches Paket kann eine oder mehrere Message-IDs enthalten, um damit das Löschen von einem oder mehreren Artikeln veranlassen. Die Sender-ID sollte von der normalen Sender-ID des Hosts abweichen und in der Form "Sender-ID-cancel" sein. Also z.B. "blackbush-cancel" oder "news.xlink.net-cancel". Dies ist ist nötig, um das Akzeptieren von *Cancel* Nachrichten besser zu regulieren.

Daten gleichmässiger versenden *Mcntp* versendet die Daten so schnell es geht. Dies kann unter Umständen sehr unregelmäßig sein wenn z.B. zwei große Artikel nacheinander kommen, dann ein paar Sekunden Pause sind, und dann ein kleiner Artikel kommt. mcxmit müßte in der Lage sein, herauszufinden, wie groß die durchschnittliche Datenrate ist, um dann nach Artikeln, die diese Rate überschreiten, eine kleine Pause einzulegen, um die Rate zu "glätten".

7.2 Ausblick

Mcntp wurde erstmals auf der Jenc8 (Joint European Networking Conference 8) in Edinburgh vorgestellt. Ein Nachdruck des zu dieser Konferenz eingereichten Paper bedindet sich im Anhang. Das Interesse an *Mcntp* ist groß und es steht zu erwarten, daß *Mcntp* nach der allgemeinen Freigabe viel eingesetzt wird.

Das Protokoll von *Mcntp* wurde der IETF (siehe auch Abschnitt 2.5 auf Seite 22) als experimentellen Standard vorgeschlagen, indem ein Internet-Draft eingereicht wurde. Ein Abdruck der aktuellen Version des Drafts befindet sich ebenfalls im Anhang. Es steht zu hoffen, daß *Mcntp* als experimentelles Protokoll angenommen und als RFC veröffentlicht wird, um es einer breiten Öffentlichkeit zugänglich zu machen und die Weiterentwicklung zu fördern.

Der Einsatz von *Mcntp* im Backbone von Xlink wird weiter ausgebaut werden, um eine weitere Entlastung der Leitungen zu erreichen. Dazu wird *Mcntp* auf dem nächsten POP-Treffen den POPs allgemein vorgestellt werden.

In Zusammenarbeit mit Orion Network Systems Inc., einem amerikanischen Satellitenprovider, wird geprüft , inwieweit *Mcntp* für die Newsverteilung via Satellit geeignet ist.

7.3 Weitere Informationen

Die Entwicklung von *Mcntp* wird fortgesetzt. Um aktuelle Informationen zu erhalten, bestehen die folgenden Adressen:

- Weitere Informationen über *Mcntp* sind im WorldWideWeb unter der URL
  http://www.pilhuhn.de/mcntp/
  verfügbar.
- Es existiert auch eine Mailingliste zu *Mcntp* – zur Teilnahme bitte eine EMail an
  <majordomo@pilhuhn.de>
  senden. Diese Mail sollte sollte die Zeile

  ```
  subscribe mcntp
  ```

  enthalten. Sollen die Listenmails an eine andere Adresse gehen, ist diese Adresse ebenfalls anzugeben (z.B.):

  ```
  subscribe mcntp joe.user@some.do.main
  ```

- Vorschläge und Fehlerberichte bitte per EMail an <mcntp-bugs@pilhuhn.de> senden.

7.4 Danksagungen

Ich möchte den folgenden Personen und Firmen Dank sagen, die mich direkt oder indirekt bei der Arbeit an *Mcntp* unterstützt haben:

- Meine Eltern, die mir das Studium ermöglicht haben.
- Prof. G. Schneider, bei dem ich diese Diplomarbeit anfertigen konnte und der mir die Anmeldegebühr zur Jenc8 Konferenz zugeschossen hat.
- Sabine Dolderer, die mich während dieser Arbeit betreut und diese Ausarbeitung korrekturgelesen hat.
- Markus Lauf, der mich ebenfalls betreut hat.
- Die Firma NTG/Xlink, die mir den Flug zur Jenc8 Konferenz bezahlt hat. und auf deren Rechnern und Backbone ich *Mcntp* testen konnte.
- Matthias Urlichs, André Beck, Jörg Mann und Ralf Gebhart, die *Mcntp* getestet haben.
- Andrea Warzel und Wolfgang Tremmel, diese Ausarbeitung korrekturgelesen haben.

A Anhang

A.1 Abdruck des zu Jenc8 eingerichten Papers

Transport of NetNews via IP-Multicast

Heiko W.Rupp
<hwr@pilhuhn.de>
18.02.1997

Abstract

The Bandwidth consumed by NetNews is growing steadily. While some time ago the rule of thumb was that it will double every year it seems now less dramatic, but still increasing. Even if NetNews used local caches since it starting (news spool at the provider) and a good flood fill algorithm for distribution, the bandwidth use of the physical links is still not optimal. Article transport via IP-Multicast can be a good way to improve link usage.

What is now

On the Internet distribution of NetNews is accomplished on a store and forward base. A host that gets an article, stores it locally in its databases and then offers it via NNTP[KL86] to all its configured neighbours that might be interested in the article. Neighbours that already have the article refuse to accept them so that they won't get transferred to a remote twice. This is a efficient way to distribute the articles, but it doesn't take into account that for offering and especially transfer of articles these often traverse the same physical link multiples times and thus imposing there a load that is much bigger than it has to be.

One way to oppose this load is to install servers on all nodes where backbone links meet and just let these servers talk to each other. The disadvantage of this system is that additional servers are needed which cost a lot of money.

What can be

Nowadays most backbone routers are already able to distribute multicast data, so one can use this infrastructure to get a mesh of distribution where articles don't cross links multiple times. If a site wants to get some newsgroups it just joins the appropriate multicast group. Most modern operating systems follow RFC 1112 [Dee89] and so have the prerequisite to send and receive news via multicast.

Kurt Lidl, Josh Osborne and Joseph Malcolm already described such a mechanism of distribution in [LOM94], but this one never got into production mode as it was too slow; also code is not free. There were also no provisions for spooling of articles if the remote server is down nor for compression of the articles before transmission.

There are basically two approaches to get NetNews over IP-multicast working:

Reliable multicast This one provides a way of distribution that, like TCP, guarantees you (within some bounds) that the data you are sending off arrives at the receivers completely. For this to work you need either some additional hard- or software that provides the fault tolerance[Hof94, SDW92]. Or, if you let all receivers send their acknowledge packets directly to the multicast sender, you will get network and performance problems there (known as the implosion problem) [LS95].

Unreliable multicast Here IP packets with some payload are just put on the wire and flow to the receiver. Packets may get lost as depending on line quality and may arrive in wrong order. As packet loss rate is monitored in daily use and kept on a low level, this is no problem in practice[1].

The way to go is with unreliable multicast for the following reasons:

- Wrong article order doesn't matter as it is not important for news articles to arrive in order. Reader software takes care of that and different users want different sorting of articles anyway.
- There is no need for additional hard- or software which reduces costs of machines and administration.
- It doesn't matter much if some articles get lost as standard NNTP is a good way to deliver this lost articles reliably.
- There is no implosion problem.
- With most articles fitting in just one UDP packet (see below) the overhead of sending acknowledges and or keeping local copies (in the routers or dedicated hops) until all articles are reliably delivered is too high.
- Can be used on the existing MBONE [Dee93].

[1] if packet loss happens due to link overload then a multicast feed instead of several unicast feeds can even improve the situation

After all this considerations, the following is done at the sender: For each article that is appropriate, compress it if you want, put it into a UDP packet, add some checksum and send it off the net. After some time offer the article to the remote via NNTP. At the receiver the packet is decoded and offered to the (local) news system via NNTP. If the local server is not available spool the article to a local disk. Article compression at the sender is done with the freely available Zlib [LD96].

Some data

Research has shown that over 90% of all articles are less than 64 kBytes in size[2] so that they fit well into the 64 kByte limit of UDP packets.

With a very alpha version of the code it was possible to get between 50 and 80 % of the articles via multicast even if a real time feed with *nntplink* was offering the same articles in parallel. If this NNTP feed is delayed for some time (e.g. 30 seconds) then the count of successfully transferred articles via multicast will even increase.

Conclusion

Transport of NetNews via multicast is a good way to go. If it is done ”right”, then it can even compete with the speed of fast classical transport mechanisms as *nntplink* while still saving bandwidth.

About the author

I am a undergraduate student of computer science at University of Karlsruhe, Germany. *Transport of NetNews via IP-Multicast* is my diploma thesis. To earn my cookies I am working at German ISP NTG/Xlink[3] as administrator of Netnews.
The address for contact is:

Heiko W.Rupp
Gerwigstr.5
D-76131 Karlsruhe
Germany
Tel.: +49 721 966 1524
Fax: +49 721 661937
Email: hwr@pilhuhn.de

Sponsors are welcome!

References

References

[Dee89] Steve Deering. *RFC 1112 – Host extensions for IP multicasting.*, August 1989. replaces RFC 1054 and RFC 988.

[Dee93] S. Deering. *MBone: The Multicast Backbone*, M”arz 1993. CERFnet Seminar.

[Hof94] M. Hofmann. *Zuverl”assige Kommunikation in heterogenen Netzen*, August 1994. Diplomarbeit am Institut f”ur Telematik, Universit”at Karlsruhe.

[KL86] Brian Kantor and P. Lapsley. *Network News Transfer Protocol.*, Februar 1986.

[LD96] J. Gailly L. Deutsch. *ZLIB Compressed Data Format Specification version 3.3*, Mai 1996.

[LOM94] Kurt Lidl, Josh Osborne, and Joseph Malcolm. Drinking from the firehose: Multicast usenet news. Technical report, UUNET Technologies Inc., Januar 1994.

[LS95] John C. Lin and Paul Sanjoy. *RMTP: A Reliable Multicast Transport Protocol*, Februar 1995.

[SDW92] W. T. Strayer, D.J. Dmepsey, and B.C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, Reading, Mass., USA, 1992.

[2] http://www.xlink.net/ hwr/histo/
[3] http://www.xlink.net/

A.2 Abdruck des eingereichten Internet Drafts

```
INTERNET DRAFT          EXPIRES MARCH 1998      INTERNET DRAFT

Network Working Group                                    Heiko W.Rupp
Request for Comments: nnnn
Experimental                                               16.09.1997


         A Protocol for the Transmission of Net News Articles
                        over IP multicast
                    <draft-rfced-exp-rupp-03.txt>

Status of This Memo

This document is an Internet-Draft.  Internet-Drafts are working
documents of the Internet Engineering Task Force (IETF), its
areas, and its working groups.  Note that other groups may also
distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six
months and may be updated, replaced, or obsoleted by other
documents at any time.  It is inappropriate to use Internet-
Drafts as reference material or to cite them other than as
"work in progress."

To learn the current status of any Internet-Draft, please check
the "1id-abstracts.txt" listing contained in the Internet-
Drafts Shadow Directories on ftp.is.co.za (Africa),
nic.nordu.net (Europe), munnari.oz.au (Pacific Rim),
ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited.


Abstract

   Mcntp (Multicast News Transfer Protocol) provides a way to use the IP
   multicast infrastructure to transmit NetNews articles between news
   servers. Doing so will reduce the bandwidth that is actually needed
   for transmission of articles which is mostly done via NNTP. This does
   not affect how news reading clients communicate with servers.

Overview and Rationale

   NetNews are bulk data that are produced in large quantities every day
   around the world. Distribution of NetNews on the Internet are usually
   distributed with NNTP[1]. In order to get a fast and redundant
   distribution many news servers communicate with many others, thus
   imposing a higher load on the underlying network than necessary.

   Assume the following scenario:

                             +---------  R1
                            /
          S -- A ------- B  --------  R2
                            \
                             +--------- R3

   A sender S which wants to transmit articles via NNTP to receivers
   R1...R3 will thus transmit them three times across the link from A to
   B.

   With IP Multicast[2], an efficient way to distribute datagrams to
   groups of users exists in the Internet. Thus articles would traverse
   the link A to B only once, thus reducing load on that link.

   This cannot be done with existing news transfer technology, as it is

Rupp                                                          [Page 1]
```

based on TCP[10] which cannot be multicasted. The protocol described in this memo is designed to put news articles into datagrams and distribute them via IP multicast to receivers that are interested in the specific newsgroup. For more information about NetNews, refer to [7] and [1].

Protocol overview

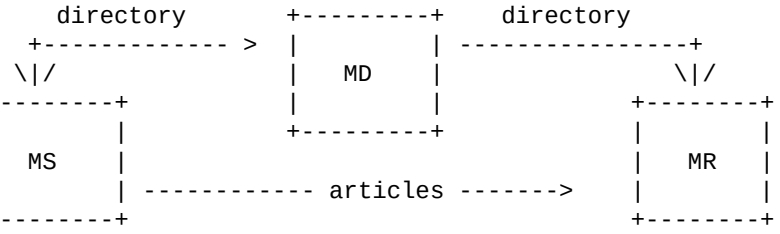
This paragraph will show how news articles are propagated with Mcntp. Basically, three parties are involved:

+ Multicast directory service, MD, coordinates the assignment between multicast and news groups.

+ A Multicast sender, MS, that sends news articles over an IP multicast infrastructure

+ A Multicast receiver, MR, gets packets from the IP multicast infrastructure and processes them further.

So this can be seen as follows:



MS and MR can be implemented into existing news server software, or can be implemented as separate processes that communicate with the news servers (e.g. via NNTP); this does not matter to the protocol.

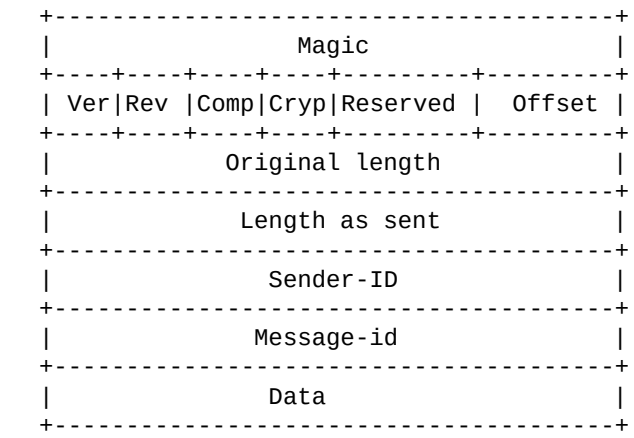
MD can either be implemented within MS, or as separate processes that communicate with each other. A practical way is to have on MD per sender host so that communication between MD and MS is fast and reliable, while not too many resources are needed.

The protocol itself consist of two parts that will be presented in the next two chapters -- Distribution of articles and the directory service.

Packet format -- Distribution of articles

To send articles via IP multicast, they have to be encapsulated into

UDP packets. The following diagram shows how this can be done:



All entries are in network byte order. The fields have the following meaning and types:

+ Magic (32-bit): The String ‘‘McNt’’

+ Ver (4-bit): Protocol version -- currently 1

+ Rev (4-bit): Protocol revision -- currently 1

+ Comp (4-bit): Compression method used. Currently are only 2 methods defined:

  0 Article is not compressed

  1 Article is compressed via zlib [8]

+ Cryp (4-bit): Encryption method used. See below

+ Reserved (8-bit): Reserved for future extensions.

+ Offset (8-bit): Offset of article data from packet start

+ Original length (32-bit): Length of article before compression, encryption and signing.

+ Length as sent (including digital signature) (32-bit): Size of the ‘‘Data’’ field (see below)

+ Sender-ID: Identification of the sender host terminated by a

null byte (see below).

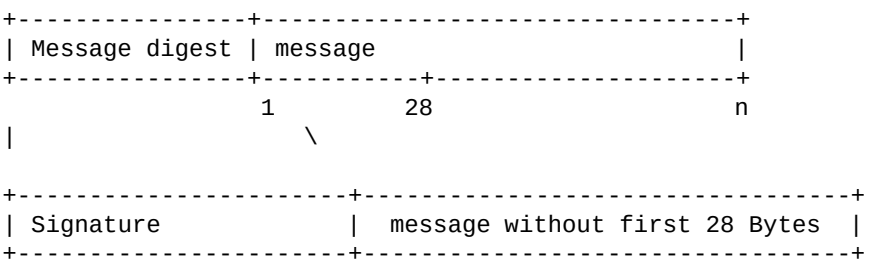
+ Message-ID: The message id of the article in the form it is defined in RFC 1036 [7], terminated by a null byte.

+ Data: The signed article data after possible compression and encryption.

This memo does not specify a encryption method for the case that the field ''Crypt'' is set to anything other than 0; the involved parties (i.e. the senders of encrypted news and their receivers) have to agree on a method they want to use. If encryption and compression is used then the article data is first to be compressed and then the result to be encrypted.

All articles must be signed before sending them off the net. This is accomplished by running the RIPEMD-160 message digest [11] algorithm over the (possibly compressed and encrypted) article and then RSA-encrypting the message digest with the private key that is suitable for sender-id. The receiver decrypts the signature of the article with the public key of sender-id and runs RIPEMD-160 over the data to see if it has been altered on the way. An article with an invalid signature or a non matching message digest has to be thrown away. The sender-id can be the path entry or the hostname of the sending site; there can also be more than one key pair per site e.g. to have different keys for different newsgroups. The sender-id has to be treated in a case independent manner.

Encryption of the message digest is done the following way. The 20 Bytes RIPEMD-160 message digest and the first 28 bytes of the (possibly compressed and encrypted) message are tacked together to form a 48 Bytes buffer. This buffer is then encrypted with the right RSA private key and prepended to the original message without the first 28 bytes:



To send an article off, it is encapsulated and then just sent to the appropriate multicast group. There is no feedback from the receiver to the sender when an article is received.

Directory service

In order to get a relation between newsgroups and multicast groups, a directory service exists; this has been referenced as MD above. When a sender MS wants to propagate a news group, it asks the directory service for a multicast group it can use to distribute articles, waits for the reply, and starts to send. The directory server registers this group in its tables and periodically distributes this table over IP multicast. For this purpose, the multicast group ''mcntp-directory.mcast.net'' has been officially been assigned by the IANA. The UDP port which announcements are sent to, has officially been assigned by the IANA as UDP port number 5418 with the name ''mcntp''.

Announcements should not be sent too often to keep traffic low, but often enough that new receivers don't have to wait to long to be able to receive articles. Once a minute is assumed to be a good value here. Announcements can be sent less often if they are transmitted immediately after a change in the directory.

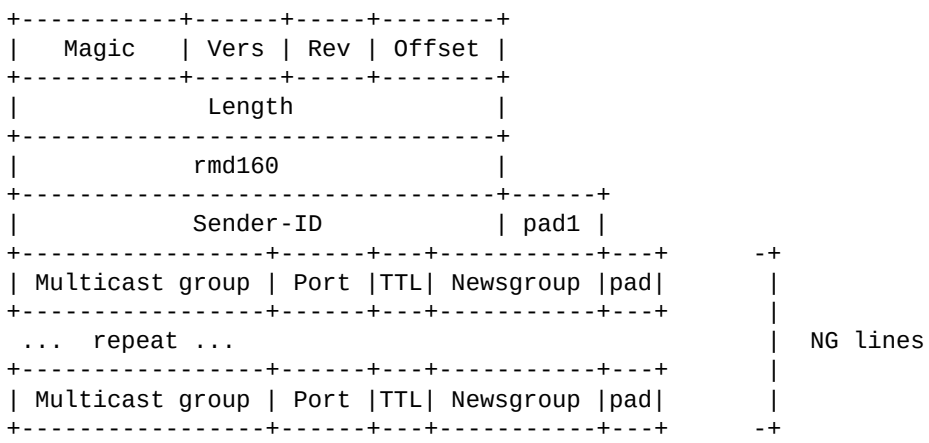
If more than one directory server is involved (e.g. if there is more than one sender site), the directory servers have to listen to announcement packets on ''mcntp-directory.mcast.net''. If it does not receive a packet after five times the waiting period (e.g., five minutes) it can consider itself alone on the net and can choose the multicast groups as it wishes. See below on usage scenarios which further explain this.

Groups that are local to an organisation (e.g. an ISP) or should stay within their bounds, must be transported within the range of the administratively scoped multicast addresses [12].

When a receiver (MR) wants to receive a newsgroup, it listens on ''mcntp-directory.mcast.net'' for announcements, parses them, and then joins the appropriate multicast groups.

Multicast groups that are no longer in use (e.g. because the sender has stopped working) must be removed from the announcement.

The format of those announcement packets is:



All numbers are in network byte order. The fields have the following meaning and types:

+ Magic (16-bit): The Bytes 0xabba.

+ Vers (4-bit): Protocol version (see below).

+ Rev (4-bit): Protocol revision (currently 1).

+ Offset (8-bit): Offset of NG-lines from packet start.

+ Length (32-bit): Total packet length.

+ rmd160 (160-bit): RIPEMD-160 message digest over the rest of the packet.

+ Sender-ID : Identification of sender host, terminated by a null byte (see below).

+ Pad1: Padding to next 4-Byte boundary filled with null bytes.

+ Multicast group (32-bit *): The associated multicast group.

+ Port (16-bit): UDP Port to use for this group.

+ TTL (8-bit): Time to live for multicast packets.

+ Newsgroup: Name of the Newsgroup(s), terminated by a null

byte. See also below.

\+ Pad: Padding of the string to the next 4-bytes boundary filled with null bytes.

The protocol version (Vers) is currently 1 for IPv4 and 11 for IPv6. The multicast group field (*) is 32 bit in size for IPv4 and 128-bit for IPv6 in size.

The length field is 32 bit in size to support IPv6 jumbo datagrams.

The sender-ID is normally the fully qualified domain name of the hosts that sends the announcement. As is common practice with NetNews, this can also be the (possibly shorter) entry that the host puts in the ‘‘Path:’’ header when an article passes through it. This entry has to be treated in a case independent manner.

The rmd160 is computed over the sender-id field and all lines with newsgroup to multicast group relations in the packet with the RIPEMD-160 message digest algorithm.

The lines with newsgroup to multicast group relations are repeated as often as needed to announce all groups. The TTL can be used by clients to find out if packets that come from this source can reach them, or if the sender is too far away. Note that all entries have to fit into one UDP packet.

The sender-id and the newsgroups entries are padded to the next 4-bytes boundary in order to make processing easier.

TTL values of articles have to be chosen, especially for use on the MBONE, in a way that newsgroups that are only of local relevance (e.g. campus groups or groups local to a town) are not distributed out of their normal distribution area. As already mentioned above, articles that are only of a local meaning or of local relevance, must be distributed within the administratively scoped group range [12].

The relation between multicast and newsgroups can range from one multicast group per newsgroup over one multicast group per news hierarchy (e.g. comp.*) to all articles in only one group. As current implementations of kernels and routers get inefficient with too many multicast groups, the use of one multicast group per newsgroup is deprecated.

Reliability Considerations

As UDP is a unreliable service, provisions for reliable distribution of articles are needed. There exist some approaches to reliable multicast (XTP [4], KLG [5] RMTP [6] and others) which all suffer from some problem or other. Specifically, additional hard- or software is needed and usually requires kernel modification.

As there is already a reliable transport of NetNews via NNTP, there is no need for a reliable transport via IP multicast: articles need not be in order, so it is no problem if one is missing in the multicast. Since articles need not arrive in order, lost or missing articles can easily be transmitted via an additional NNTP feed.

As UDP packets can be at maximum 64kBytes in size and every Mcntp packet has to fit in one UDP packet, there is no provision given to distribute news articles larger than about 63kBytes in size (other than compressing them). This does not matter much in practise as recent research has shown that more than 95% off all news articles are smaller 64kBytes [9]. The remaining 5% can still be transferred via NNTP. Some hosts may have problems in receiving UDP packets as large as 64kBytes, so in practical use article sizes of 16kBytes would be appropriate. These are still over 90% of all articles.

Usage Scenarios

These scenarios show how mcntp can be used in daily use. The main difference between local and MBone wide usage are the multicast groups that are used for distribution as stated above.

For a local use within an organisation there could be one central sending site that redistributes all news articles it receives via mcntp. No further action is needed.

When more than one directory server (MD) gets involved, directory servers must wait on startup for announcement packets from other MD processes, register the contained groups in its tables and make decisions involving that tables. Decisions can be divided into the following:

Use

If the group in which the sender (MS) wants to send is already distributed over multicast, then the articles are distributed in the existing group else a new multicast group is used. For example: if de.* is already distributed over multicast group a.b.c.d then use that group.

New

Always create own multicast groups that don’t clash with the ones that are already existing. For example: if comp.* is already distributed on group a.b.c.e and the sender (MS) wants to distribute comp.foo, don’t use group a.b.d.e, but create a new one.

Standby

Only send articles for a specified newsgroup when no one other is doing it. This can be used to implement backup functionality. For example: Sender A is sending comp.*. If now a directory packet arrives at site B, which no longer has comp.* in it, B can start to send comp.*. When it sees again announcements from A, then it stops the distribution of comp.*.

For use in an environment, where multiple organisations are involved (e.g. on the MBone), the following could deployed: everyone that is participating utilizes the ‘‘use’’ method described above. It only sends articles that are locally produced (e.g. customers) and which are not distributed via mcntp by another site. No articles received from news peers should be distributed that way. After some delay (at least 10 seconds), articles which are distributed via mcntp are offered to peers over nntp as usual. The set of groups that is distributed must be negotiated between the involved organisations. With the current Usenet groups this could be:

- rec.*
- comp.*,news.*,gnu.*
- talk.*,misc.*
- humanities.*,sci.*,soc.*
- alt.*

Summary

The distribution of NetNews articles via IP multicast can be a way to decrease the network bandwidth used to distribute them. Articles are delivered fast via a nonreliable protocol; later, the holes are filled via a reliable, already existing protocol. Compression of articles can further reduce the network Load. With encryption private news groups can be established on a public IP multicast infrastructure. A prototype of a reference implementation already shows that Mcntp is fast and can be used as an alternative to classical transports. The use of zlib for compressing articles shows a reduction of transferred volume (including protocol headers) to about 65% of the original article volumes..

Security Considerations

With the classical NNTP based distribution, every host on the path of an article keeps track of it in the logfiles, making it possible to find the sender of forged or abusive articles with the aid of the administrators of the newshosts along the path. For the distribution of NetNews over IP multicast, this is no longer true: routers don't log packets flowing by and as the sender address of IP packets can be forged, a sender can't be traced. This fact can be used to inject forged news articles without being traceable.

To prevent the unnoticed injection of articles, a mcntp receiver only accepts articles from senders that it trusts. This trust is build by digitally signing the article with the private key of the sender and verifying the signature at the receiver site. Receivers have to accept only articles with good signatures

The RIPEMD-160 message digest algorithm has been chosen, as it is more secure than MD5 while still being fast enough. The RSA encryption algorithm has been chosen as there exist reference implementations for usage inside US (from RSA Inc.) and outside (rsaeuro by J.S.A.Kapp).

The key size for the RSA algorithm must be at least 512 bit in size to prevent cracking of the key.

References

[1] RFC 977 -- B. Kantor, P. Lapsley, "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News".

[2] RFC 1112 -- S. Deering, "Host extensions for IP multicasting", 08/01/1989.

[3] RFC 768 -- J. Postel, "User Datagram Protocol", 08/28/1980.

[4] XTP -- W. T. Strayer, D.J. Dmepsey, B.C. Weaver, "XTP: The Xpress Transfer Protocol", Addison-Wesley

[5] KLG -- M. Hofmann, "Zuverlaessige Kommunikation in heterogenen Netzen", Thesis at "Institut fuer Telematik, CS Dept. Univ Karlsruhe"

[6] RMTP -- Lin, John C., Paul Sanjoy, "RMTP: A Reliable Multicast Transport Protocol".

[7] RFC 1036 -- M. Horton, R. Adams, "Standard for interchange of USENET messages", 12/01/1987.

[8] RFC 1950 -- L. Deutsch, J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", 05/23/1996.

[9] http://www.xlink.net/~hwr/histo/ -- Some Statistics about size distribution of NetNews

[10] RFC 793 -- J. Postel, "Transmission Control Protocol", 09/01/1981.

[11] H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD" 04/18/1996. An earlier version appeared in "Fast Software Encryption,LNCS 1039" Springer Verlag, 1996, pp. 71-82.

[12] [draft-ietf-mboned-admin-ip-space-04.txt/number of rfc] David Meyer, "Administratively Scoped IP Multicast", [date of rfc]

Author's Address

Heiko W.Rupp
Gerwigstr. 5
D-76131 Karlsruhe

Phone: +49 721 9661524

EMail: hwr@pilhuhn.de

Literatur

[BF93] N. Borenstein and N. Freed. *RFC 1521 – MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.*, September 1993. Ersetzt RFC 1341.

[Bra96] S. Bradner. *RFC 2026 – The Internet Standards Process – Revision 3*, Oktober 1996.

[Cro82] D. Crocker. *RFC 822 – Standard for the format of ARPA Internet text messages*, August 1982.

[DBP96] H. Dobbertin, A. Bosselaers, and B. Preneel. Ripemd-160: A strengthened version of ripemd. Technical report, April 1996. An earlier version appeared in Fast Software Encryption,LNCS 1039, Springer Verlag.

[Dee89] Steve Deering. *RFC 1112 – Host extensions for IP multicasting.*, August 1989. Ersetzt RFC 1054 und RFC 988.

[Dee93] S. Deering. *MBone: The Multicast Backbone*, März 1993. CERFnet Seminar.

[Deu96a] L. Deutsch. *RFC 1951 – DEFLATE Compressed Data Format Specification version 1.3*, Mai 1996.

[Deu96b] L. Deutsch. *RFC 1952 – GZIP file format specification version 4.3*, Mai 1996.

[FM96] N. Freed and K. et. al. Moore. *RFC 2045-2049 – Multipurpose Internet Mail Extensions (MIME)*, November 1996.

[Gib95] Mark Gibbs. *Durckblick im Netzwerk.* iwt Verlag GmbH, 1995.

[HA87] M. Horton and R. Adams. *RFC 1036 – Standard for interchange of USENET messages*, Dezember 1987.

[Har95] Mark Harrison. *The Usenet Handbook*. O'Reilly and Associates, Sebastopol, Calif., USA, Mai 1995.

[Hof94] M. Hofmann. *Zuverlässige Kommunikation in heterogenen Netzen*, August 1994. Diplomarbeit am Institut für Telematik, Universität Karlsruhe.

[Kal92] B. Kaliski. *RFC 1319 – The MD2 Message-Digest Algorithm*, April 1992.

[KL86] Brian Kantor and P. Lapsley. *Network News Transfer Protocol.*, Februar 1986.

[Kro94] Ed Krol. *The wohle Internet*, chapter 13. O'Reilly and Associates, Sebastopol, Calif., USA, 2 edition, 1994.

[Kum96] Vinay Kumar. *MBone - Interactive Multimedia on the Internet.* New Riders, 1996.

[Kus94] Michael Kuschke. Vervierfacht. *iX Multiuser Magazin*, 94(10):132–136, Oktober 1994.

[LD96] J. Gailly L. Deutsch. *RFC 1950 – ZLIB Compressed Data Format Specification version 3.3*, Mai 1996.

[LS95] John C. Lin and Paul Sanjoy. *RMTP: A Reliable Multicast Transport Protocol*, Februar 1995.

[Mey97] David Meyer. Administratively scoped ip multicast "draft-ietf-mboned-admin-ip-space", 1997. This is a working draft of the IETF MBONED Working Group.

[Moy94] J. Moy. *RFC 1584 – Multicast Extensions to OSPF*, Miärz 1994.

[NF97] K. Moore N. Freed. *RFC 2184 – MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations.*, August 1997. Aktualisiert RFC 2045.

[OT92] Tim O'Reilly and Grace Todino. *Managing UUCP and Usenet*, chapter 7 ff. O'Reilly and Associates, Sebastopol, Calif., USA, Januar 1992.

[P.L92] G.Krüger P.Lockemann. *Kommunikation und Datenhaltung*. Universität Karlsruhe, Fakultät für Informatik, Am Fasanengarten 5, D-76128 Karlsruhe, 1992.

[Pos80] Jonathan B. Postel. *RFC 768 – User Datagram Protocol*, August 1980.

[Pos81a] Jonathan B. Postel. *RFC 791 – Internet Protocol*, September 1981. Darpa Internet Protocol Specification.

[Pos81b] Jonathan B. Postel. *RFC 792 – Internet Control Message Protocol*, September 1981.

[Pos81c] Jonathan B. Postel. *RFC 793 – Transmission Control Protocol*, September 1981.

[Pos82] Jonathan B. Postel. *RFC 821 – Simple Mail Transfer Protocol*, August 1982.

[Pos93] Jonathan B. Postel. *RFC 1543 – Instructions to RFC Authors*, Oktober 1993.

[Pus96] T. Pusateri. Distance vector multicast routing protocol "draft-ietf-idmr-dvmrp-v3", 1996. This is a working draft of the IETF.

[Riv78] Adelman Rivest, Shamir. *A Method for Optaining Digital Signatures and Public Key Cryptosystems*, Februar 1978. Communications of the ACM, 21, s, S.120-126.

[Riv92a] R. Rivest. *RFC 1320 – The MD4 Message-Digest Algorithm*, April 1992.

[Riv92b] R. Rivest. *RFC 1321 – The MD5 Message-Digest Algorithm*, April 1992.

[Rup97] Heiko W. Rupp. Transport of netnews via ip-multicast. Technical report, Universität Karlsruhe und NTG/Xlink, Karlsruhe, hwr@pilhuhn.de, Februar 1997. URL: http://www-old.xlink.net/∼hwr/-jenc8.ps.

[SDW92] W. T. Strayer, D.J. Dmepsey, and B.C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, Reading, Mass., USA, 1992.

[Sed88] Robert Sedgewick. *Algorithms*, chapter 22. Addison-Wesley, Reading, Mass., USA, 1988.

[SRL96] Kevin Savetz, Neil Randall, and Yves Lepage. *MBONE: Multicasting tomorrow's Internet*. IDG Books, 1996.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Mass., USA, 1994.

[UCB94a] Computer Science Resarch Group U C Berkeley. *4.4 BSD Programmer's References Manual*. O'Reilly and Associates, Sebastopol, Calif., USA, 1994.

[UCB94b] Computer Science Resarch Group U C Berkeley. *4.4 BSD Programmer's Supplementary Documents*, chapter 21. O'Reilly and Associates, Sebastopol, Calif., USA, 1994.

[WD76] M. Hellman W. Diffie. *New Directions in Cryptography*, November 1976. IEEE Transactions on Information Theory, IT22, No.6.

[WPD88] D. Waitzman, C. Partridge, and Steeve Deering. *RFC 1075 – Distance Vector Multicast Routing Protocol*, November 1988.

Index

Glossar

Big-Endian Zahlendarstellung in Prozessoren, bei denen das höchstwertige Byte eines Wortes in der ersten von mehreren, aufeinanderfolgenden Speicherzellen abgelegt ist. Beispiel für Big-Endian Maschinen sind: Motorola 680x0, IBM AIX oder Sun Sparc. Siehe auch: Little-Endian.

CIDR Classles Internet Domain Routing – Routingschema, das ohne die festen Klasseneinteilungen auskommt und die Routen je nach Anforderung zu größeren Blöcken zusammenfasst.

Crossposting Ein Newsartikel, der mehrere Einträge in der *Newsgroups* Zeile hat. Dieser Artikel erscheint in all den angegebenen Gruppen, wird aber nur einmal zu einem Zielsystem transportiert und dort auch nur einmal im Dateisystem abgelegt.

Datagramm ein einzelnes Datenpaket, vergleichbar mit einem Brief im gelben Postwesen. Siehe auch IP.

Datex-P Paketvermittelter Dienst der Deutschen Post AG, der nach dem X.25 Standard der CCITT arbeitet.

dotted-quad Format, in dem IP-Nummern geschrieben werden und das aus vier einzelnen Bytes besteht, die durch Punkte getrennt sind: a.b.c.d. Beispiel: `193.141.40.8`

IANA Abkürzung für Internet Assigned Numbers Authority. Eine Organisation im Internet, die zentral die Vergabe von IP- und Portnummern und auch Multicastadressen regelt. Siehe auch Abschnitt 2.6.

IETF Internet Engineering Task Force. Die IETF ist die Organisation im Internet, die sich mit der Weiterentwicklung und Standardisierung des Internets beschäftigt.

Implosion Für jedes versandte Datenpaket werden viele Antwortpakete erhalten.

INN Ein populärer NTA, der von Rich Salz entwickelt wurde.

IP InternetProtokoll – Protokoll zur Rechnerkommunikation auf Schicht 3 des ISO Modells.

ISP Abkürzung für Internetserviceprovider.

Link (oder auch Hyperlink). Ein Teil in einem Text, der angewählt werden kann, um einem Verweis zu einem neuen Dokument zu folgen. Wird in der Sprache HTML verwendet.

Little-Endian Zahlendarstellung in Prozessoren, bei denen das niederwertigste Byte eines Wortes in der ersten von mehreren, aufeinanderfolgenden Speicherzellen abgelegt ist. Beispiel für Little-Endian Maschinen sind: Intel ix86 und Digital Vax. Siehe auch: Big-Endian.

Mrouter Ein Router, der auch multicastfähig ist.

NetNews Die NetNews sind eine Art verteiltes, virtuelles schwarzes Brett, die nach Themenbereichen untergliedert sind. Jeder Benutzer der NetNews kann neue Artikel an dieses virtuelle Brett heften.

Network Byte Order Zahlendarstellung auf den Internet. Die Network Byte Order entspricht der Zahlendarstellung auf Little-Endian Maschinen.

Netzmaske Eine Maske, die mit der IP-Nummer durch ein logisches *und* verknüpft wird. Der Teil in der Netzmaske, der auf Eins steht gibt dabei den Netzteil und der, der auf Null steht, den Hostteil der Adresse an.

NNTP Abkürzung für NetNews Transfer System. NNTP ist in RFC977[KL86] definiert.

NTA Abkürzung für Newstransferagent. Analog zu MTA für Messagetransferagent.

NUA Newsuseragent. Programm mit dem der Benutzer Newsartikel lesen und erzeugen kann.

POP
1. Point of Presence. Lokaler Partner/Zugangspunkt eines Internetserviceproviders, ISP.
2. Post Office Protocol. Ein Protokoll, um elektronische Post zu übertragen.

Port Ein Port im Zusammenhang mit IP ist ein Kommunikationsendpunkt, der folgende drei Eigenschaften hat: IP-Nummer, Portnummer, Protokoll.

Provider Diensteanbeiter. Kurzform für Internetserviceprovider (ISP).

Router Ein Rechner, der Datenpakete auf ISO-Schicht 3 weiterleitet.

Spool Der Spool ist der Bereich im Dateisystem, in dem Daten zur Abholung bereitgehalten werden. Im Artikelspool werden z.B. die Newsartikel zwischengespeichert.

TCP Transmission Control Protocol – verbindunsorientiertes Protokoll, das auf IP aufsetzt.

Telnet
1. Ein "Terminalemulationsprotokoll", welches erlaubt, sich auf einem anderen Rechner im Internet einzuloggen
2. Eine Applikation, die das Telnetprotokoll benutzt, um sich auf einem anderen Rechner einzuloggen.

Tunnel Eine Verbindung zwischen zwei Endpunkten. Ein Tunnel verbindet z.B. zwei multicastfähige Inseln in einer Umgebung, die nicht multicastfähig ist.

Usenet Die Gemeinschaft der Systeme, die miteinander NetNews austauschen.

WWW World Wide Web, auch W^3 genannt. W^3 ist ein System von verteilten Hypertextdatenbanken. Für Literatur siehe z.B. [Kro94]. Das WWW benutzt zur Kommunikation HTTP; Seiten werden in HTML geschrieben.