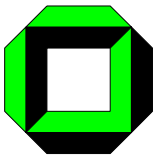

Universität Karlsruhe
Fakultät für Informatik
Institut für Telematik
76128 Karlsruhe

Architektur vernetzter Systeme

Seminar Sommersemester 2001

Herausgeber:
Marc Bechler
Verena Kahmann
Yuhong Li
Prof. Dr. L. Wolf

Universität Karlsruhe
Institut für Telematik

Interner Bericht 2001-11
ISSN 1432-7864

Inhaltsverzeichnis

Vorwort

Das Seminar *Architektur vernetzter Systeme* beschäftigt sich dieses Semester mit verschiedenen Forschungsgebieten. Zunächst ist der Bereich der *Aktiven Netze* zu nennen, in dem seit einigen Jahren an einem ganz neuen Konzept zur Vernetzung von Systemen gearbeitet wird, welches die Bearbeitung von Daten innerhalb des Netzes ermöglicht. *Kommunikationsarchitekturen im Multimedia-Bereich* stellen ein weiteres Themengebiet des Seminars dar. In diesen Bereich sind die Beiträge zum Java Media Framework und zur IP-Telefonie einzuordnen. Der Seminarband schließt mit einem Beitrag aus dem Bereich der mobilen *Ad-hoc-Netze*, welche immer noch eine besondere Herausforderung für Kommunikationsarchitekturen darstellen. Konkret werden folgende Themenstellungen behandelt:

Architektur aktiver Netze
Aktive Netze brechen mit der Tradition herkömmlicher (passiver) Netze, keinen Zugriff auf die Nutzdaten der Anwender haben zu dürfen. In aktiven Netzen erhält der Benutzer die Möglichkeit, seine Nutzdaten innerhalb des Netzes durch individuelle Programme bearbeiten zu lassen. Auch die Wegefindung kann in aktiven Netzen vom Nutzer bzw. der Anwendung beeinflußt werden. Die Netzwerkinfrastruktur läßt sich, in anderen Worten ausgedrückt, schnell an individuelle Anforderungen anpassen (Flexibilität). Dies ist der entscheidende Vorteil aktiver Netze. Weitere Anforderungen, wie die Leistungsfähigkeit und die Sicherheit, schränken die Flexibilität wieder ein. Deshalb existieren zahlreiche Architekturen, die mit unterschiedlichen Methoden versuchen, den besten Kompromiß zu finden. Einige davon werden in dieser Ausarbeitung vorgestellt und hinsichtlich dieser Kriterien miteinander verglichen. Sie basieren auf zwei verschiedenen grundlegenden Rahmenarchitekturen: aktive Pakete und aktive Netzknoten.

Ressourcenverwaltung in aktiven Netzen
In aktiven Netzen können die Benutzer die Funktionsweise des Netzes aktiv beeinflussen, indem sie Programme auf Netzknoten einbringen. Da diese Programme auf sehr heterogenen Netzknoten ausgeführt werden sollen, stellt sich die Frage, wie man die Verwaltung und den Zugriff auf Ressourcen gewähren kann, um einerseits ein hohes Maß an Flexibilität zu erreichen und andererseits Sicherheit und Effizienz sicherzustellen. Die NodeOS Schnittstelle, die beispielsweise in Janos implementiert worden ist, stellt einen Ansatz dazu dar, der in diesem Beitrag vorgestellt wird.

Anwendungen in aktiven Netzen
Erweitert man alle, oder zumindest manche Netzwerkknoten eines Netzwerkes um zusätzliche Funktionalität, so dass diese Aufgaben in allen Schichten des OSI-Modells verrichten können, ziehen netzwerkbezogene Anwendungen großen Nutzen daraus. Die Effizienz und Leistungsfähigkeit von Netzwerkmanagement, Staukontrolle, Multicasting und Caching von Daten kann bei gleichzeitiger Verringerung der Netzlast verbessert werden. Darüber hinaus erschließen sich durch den Einsatz von aktiver Netzwerktechnologie neue Anwendungsbereiche, die auf komplexe Dienste angewiesen sind, welche im Netzinneren zur Verfügung gestellt werden. Darunter fallen mobile Software-Agenten, die die Bearbeitung auf mehreren getrennten Datenbeständen in gegenseitigem Abgleich durchführen, und die so gewonnenen Informationen in aktiven Netzwerkknoten zusammenführen.

Einführung eines neuen Dienstes in aktiven Netzen
Aktive Netze repräsentieren einen signifikanten Schritt in der Entwicklung der paketvermittelten Netze, vom traditionellen Weiterleitungsmechanismus hin zu generellen Funktionalitäten zur Unterstützung dynamischer Kontrolle und Modifikation des Netzverhaltens. Dynamische Netzdienste machen diese Innovation durch neue aktive Technologien möglich. In diesem Seminarbeitrag werden heutige Anwendungen der Dienste in aktiven Netzen vorgestellt und diskutiert, wie neue Dienste in aktiven Netzen eingeführt werden.

JMF zur Entwicklung multimedialer Anwendungen
Das Java Media Framework (JMF) ist eine API (Application Programming Interface) für die Integration von zeitabhängigen Medien in Java-Applets und Applikationen. JMF unterstützt die Erfassung, die Verarbeitung und die Präsentation von zeitabhängiger Mediendaten. Ein JMF-*Player* präsentiert die Mediendaten aus einer Datenquelle. Als ein spezieller *Player* ermöglicht der *Processor* die Steuerung der Verarbeitung der Medienströme. Der *Processor* benutzt Standard-JMF-Plug-Ins oder angepasste Plug-Ins für die Verarbeitungsvorgänge wie das Demultiplexen der Eingabeströme, das Anwenden von Effektalgorithmen auf einzelnen Spuren, die Umkodierung der Mediendaten, und das Multiplexen. Das JMF Ereignismodell ermöglicht JMF Objekten, über den aktuellen Stand der Verarbeitung zu berichten bzw. auf die Ereignisse zur Laufzeit zu reagieren.

IP-Telefonie – Standards, Anwendungen und Analyse
Seit ihrer Markteinführung im Jahre 1995 durch die israelische Firma Vocaltec hat die IP-Telefonie mit rasanten Wachstumsraten auf sich aufmerksam gemacht. Alle großen Firmen im Telekommunikationsgeschäft beschäftigen sich mit der relativ neuen Technologie und bemühen sich um eine gute Positionierung im Wettbewerb. Diese Arbeit soll einen Überblick über den breiten Themenbereich geben. Dabei werden zunächst technische Grundlagen der verbreiteten Standards sowie ihre Interoperabilität dargestellt. Weitere Themen sind mögliche Anwendungen der IP-basierten Telefonie und eine Analyse ihrer Schwächen gegenüber traditionellen Telefonnetzwerken. Im Anschluss wird die IP-Telefonie in das Marktumfeld mitsamt regulatorischer und rechtlicher Problemstellungen eingeordnet. Zuletzt erfolgt ein abschliessender Ausblick mit Zukunftsprognosen und Feldern, in denen noch Handlungsbedarf besteht.

Routing in Ad-hoc-Netzen
Ad-hoc-Netzwerke sind Netze, die spontan aufgebaut werden und somit weder über eine Backbonestruktur noch über eine zentrale Verwaltung verfügen. Sie zeichnen sich dadurch aus, dass Teilnehmer ständig ein- bzw. austreten sowie ihren Standort innerhalb des Netzwerkes wechseln. Bedingt durch diese Eigenschaften stellen sie, insbesondere was Routing anbelangt, andere Anforderungen als infrastrukturbasierte Netze. Im folgenden Text wird zunächst kurz auf die Probleme von Ad-Hoc-Netzwerken im Vergleich zu konventionellen Netzwerken eingegangen. Anschließend werden generelle Charakteristika von Routingprotokollen erwähnt. Schließlich werden einige der wichtigeren Routingprotokolle erläutert und kurz verglichen.

Bevor nun die einzelnen Ausarbeitungen der Seminarbeiträge präsentiert werden, möchten wir allen beteiligten Studierenden für ihre engagierte Mitarbeit danken, ohne die weder der Erfolg des Seminars noch die Anfertigung des vorliegenden Berichts möglich gewesen wäre.

Hierzu haben auch die nach den einzelnen Vorträgen stattfindenden Diskussionen maßgeblich beigetragen.

Karlsruhe, im Juli 2001

Marc Bechler Verena Kahmann Yuhong Li Prof. Dr. L. Wolf

Architektur aktiver Netze

Matthias Unholzer

Kurzfassung

Aktive Netze brechen mit der Tradition herkömmlicher (passiver) Netze, keinen Zugriff auf die Nutzdaten der Anwender haben zu dürfen. In aktiven Netzen erhält der Benutzer die Möglichkeit, seine Nutzdaten innerhalb des Netzes durch individuelle Programme bearbeiten zu lassen. Auch die Wegefindung kann in aktiven Netzen vom Nutzer bzw. der Anwendung beeinflußt werden. Die Netzwerkinfrastruktur läßt sich, in anderen Worten ausgedrückt, schnell an individuelle Anforderungen anpassen (Flexibilität). Dies ist der entscheidende Vorteil aktiver Netze. Weitere Anforderungen, wie die Leistungsfähigkeit und die Sicherheit, schränken die Flexibilität wieder ein. Deshalb existieren zahlreiche Architekturen, die mit unterschiedlichen Methoden versuchen, den besten Kompromiß zu finden. Einige davon werden in dieser Ausarbeitung vorgestellt und hinsichtlich dieser Kriterien miteinander verglichen. Sie basieren auf zwei verschiedenen grundlegenden Rahmenarchitekturen: aktive Pakete und aktive Netzknoten.

1 Einleitung

Die fehlerfreie und möglichst schnelle Weiterleitung von Datenpaketen sind die wichtigsten Anforderungen an ein passives Netz. Die anwendungsunterstützende Bearbeitung der Nutzdaten findet hier ausschließlich in den Endgeräten statt. Durch zunehmende Verteilung von Ressourcen und technischen Systemen (z.B. im Internet) steigt der Umfang und die Komplexität der Datenübertragung in Netzwerken überproportional an. Hieraus ergibt sich schon seit Anfang der neunziger Jahre die Notwendigkeit, verschiedene anwendungsunterstützende (nicht nur vermittlungstechnische) Funktionen innerhalb der Netze zu realisieren. Als frühe Beispiele wären hierfür Web Proxies oder Firewalls zu nennen („lead user“). Web Proxies verringern die Belastung des Netzes, indem häufig benötigte Web-Seiten zwischengespeichert werden, damit sie einerseits dem Benutzer schneller zur Verfügung stehen und andererseits nicht laufend erneut geladen werden müssen. Bei dieser Lösung muß jedoch die regelmäßige Aktualisierung der zwischengespeicherten Daten veranlaßt werden, wobei sich dabei die Frage stellt, wie oft das geschehen soll. Firewalls verhindern das Weiterleiten unerwünschter Datenpakete in lokale Netze. Aber auch hier muß manuell für einen regelmäßigen Update der relevanten Informationen gesorgt werden. Beide Beipiele sind eine erste Annäherung an aktive Netze. Man bezeichnet diese Lösungen auch als *User „pull“*.

Im wesentlichen kann der Unterschied zwischen aktiven und passiven Netzen auch anhand des des OSI-Referenzmodells erklärt werden: Bei passiven Netzen beschränkt sich der Funktionsumfang auf die vier untersten Schichten (bis Schicht 4 - Transportschicht), bei aktiven Netzen reicht er bis zu den anwendungsorientierten Schichten (bis Schicht 7 - Anwendungsschicht).

Die Möglichkeit, innerhalb von paketvermittelten Netzwerken durch benutzerspezifische Prgramme auf die Nutzlast von Datenpaketen zuzugreifen und diese zu verändern oder die Wegefindung zu beeinflussen, stellt die wirklich neue Entwicklunglinie der Netzwerktechnologie dar, wobei man dann von *Technology „push“* spricht. Zugriffe und Berechnungen erfolgen

in den Knotenpunkten (z.B. Router oder Bridges) eines aktiven Netzes. Innerhalb eines aktiven Netzes müssen nicht alle Knoten aktive Funktionen unterstützen. Treffen Datenpakete auf einen herkömmlichen passiven Knoten, werden sie nach den Verfahren des zugrundeliegenden Netzdienstes weitergeleitet. Dabei werden nur Steuerinformationen der Paketköpfe (Header) ausgewertet. (In passiven Netzen sind die Nutzdaten beim Transport von einem Endsystem zum anderen für das Netz transparent.)

Die wesentlichen Vorteile des Technology „push“ kann man nun folgendermaßen zusammenfassen:

- Schnelle Anpassungsmöglichkeit der „aktiven“ Funktionen an geänderte Anforderungen.
- Unterschiedliche „aktive“ Funktionen sollten unabhängig von der zur Verfügung stehenden Hardware realisierbar sein.
- Jeder normale Benutzer (also nicht nur Systemadministratoren) sollte in der Lage sein, auf diese „aktiven“ Funktionen zuzugreifen und sie anzupassen.

Sicherheit (Safety und Security) sowie Leistungsfähigkeit sind weitere wichtige Anforderungen an ein aktives Netz:

- Safety: Schutz vor unerwünschten Funktionen im Netz.
- Security: Schutz der Nutzdaten vor fremdem unerwünschten Zugriff.
- Leistungsfähigkeit: Aktive Programme sollten umfangreiche Funktionen zur Verfügung stellen.

Die zuletzt genannten Anforderungen stehen jedoch im Gegensatz zur geforderten Flexibilität. Ziel einer aktiven Netzarchitektur sollte es nun sein, den bestmöglichen Kompromiß zu finden. Hier sind zahlreiche Varianten denkbar. In dieser Ausarbeitung wird nur eine kleine Auswahl existierender Architekturen behandelt.

2 Rahmenarchitekturen und technische Grundlagen

Beide Rahmenarchitekturen, *aktive Pakete* (auch „capsules“ oder Kapseln genannt) oder *aktive Netzknoten* (auch „programmable switches“ oder programmierbare Netzknoten genannt), haben die gemeinsame Eigenschaft, Programme ausführen zu können, die die Nutzdaten verändern und die Wegefindung im Netz beeinflussen. Hiefür sind innerhalb des Netzes, wie schon erwähnt, besondere Netzknoten (z.B. Router oder Bridges) erforderlich, die aktive Funktionen unterstützen. Der Unterschied zwischen beiden Rahmenarchitekturen besteht darin, wo sich die auszuführenden Programme befinden. Die Rahmenarchitektur der aktiven Pakete ersetzt die in passiven Netzen vewendeten normalen Datenpakete durch Kapseln, die den benutzerspezifischen Programmcode transportieren. Bei den aktiven Netzknoten hingegen befinden sich die auszuführenden Programme bereits in den Netzknoten und werden von den ankommenden Datenpaketen aufgerufen. Dabei ist noch zu unterscheiden, auf welchem Weg der Programmcode ins Netz gelangt. Man spricht dann entweder von einer *integrierten* oder einer *diskreten* Lösung. Bei der integrierten Lösung lädt der Benutzer seinen Programmcode selbst ins Netz. Aus diesem Grund zählt die Rahmenarchitektur der aktiven Pakete grundsätzlich zur integrierten Lösung. Die Rahmenarchitektur aktiver Netzknoten kann beiden Lösungen zugeordnet werden, je nachdem, ob die Programme vom Benutzer in die Netzknoten eingespielt werden oder ob dies unabhängig vom Benutzer erfolgt (z.B. durch das Netzwerkmanagement) und damit zur diskreten Lösung zählt.

Die Aktualität und die Flexibilität der Programme kann bei der Rahmenarchitektur aktiver Pakete besser gewährleistet werden, da sie erst „bei Bedarf“ in das Netz geladen werden. Der Umfang und damit in der Regel auch die Leistungsfähigkeit der Programme ist hier jedoch eingeschränkt. Außerdem steigt das Sicherheitsrisiko, weil auf diesem Weg auch unerwünschte Programme von jedem Benutzer (da integrierte Lösung) ins Netz gelangen können. Bei der Rahmenarchitektur aktiver Netzknoten kann die Leistungsfähigkeit der Programme höher sein, da wesentlich längere Programmcodes in den Netzknoten gespeichert werden können. Geschieht hierbei das Laden der Programmcodes auf diskretem Weg, so werden auch höhere Sicherheitsanforderungen erfüllt, da die Programmcodes dann nur aus einer vertrauenswürdigen Quelle (z.B. Netzwerkmanagement) stammen.

Bei der Kombination der diskreten und integrierten Rahmenarchitekturen finden sich komplexere (also auch in der Regel längere) Programme in den Netzknoten (geladen auf diskretem oder integriertem Weg). Diese werden durch aktuelle Programmfragmente, die in aktiven Paketen enthalten sind (integrierte Lösung), ergänzt und dann erst ausgeführt.

Die Defense Advanced Research Projects Agency (DARPA) beschreibt einen prinzipiell möglichen Aufbau eines Netzknotens mit aktiver Funktionalität [Calv99]. Ein solcher Netzknoten kann beide oben beschriebenen Rahmenarchitekturen unterstützen und besteht aus drei wesentlichen Komponenten (siehe auch Abbildung 1):

1. Einem oder mehreren Execution Environments (EEs), in denen die Pogramme ausgeführt werden.
2. Dem Node Operating System (NodeOS), welches die zentrale Steuerung und Anforderung der Ressourcen des Netzknotens durchführt.
3. Den Active Applications (AAs), bei denen es sich um die eigentlichen Programme handelt, die dem Benutzer aktive Funktionalität bieten. Diese Programme können, wie oben beschrieben, auf diskretem oder integriertem Weg in den Netzknoten gelangen.

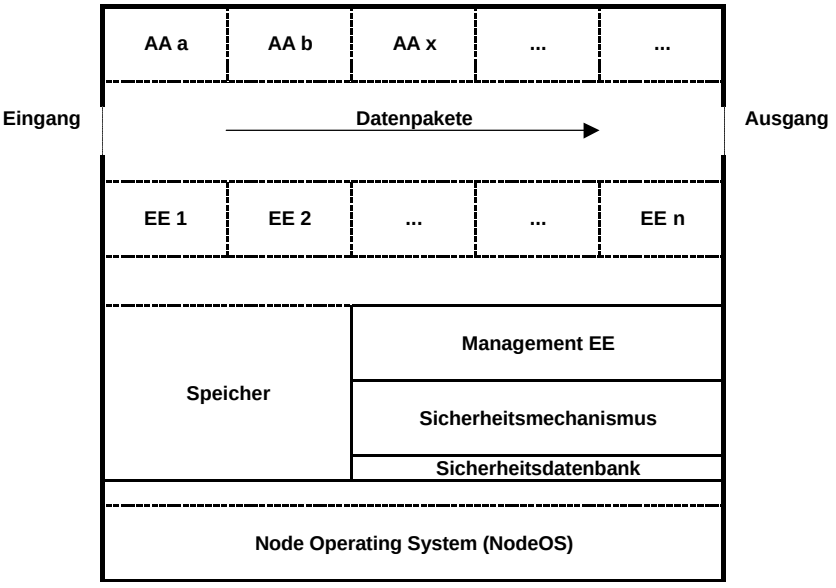

Abbildung 1: Knoten eines aktiven Netzes (Schema)

Die Management Execution Environment (Management EE) konfiguriert/aktualisiert die EEs und die Sicherheitsdatenbank. In den Knoten eintreffende Datenpakete werden identifiziert und der erforderlichen EE zugeordnet. Hierbei wird die angeforderte AA geladen. Aktive

Pakete, die ihre eigenen AAs (integrierte Programme) einbringen, werden vom Sicherheitsmechanismus anhand der Sicherheitsdatenbank überprüft, bevor sie in einer EE ausgeführt werden dürfen.

Auf diesen architektonischen und technischen Grundlagen bauen im wesentlichen die nachfolgend beschriebenen Architekturen auf. Der oben beschriebene Aufbau eines Netzknotens ist verallgemeinert und variiert bei den zahlreichen entwickelten konkreten Architketuren.

3 Architekturen

3.1 Integrierte Architekturen

Aktive Pakete enthalten Miniaturprogramme des Benutzers, die in einem oder mehreren Knoten eines aktiven Netzes ausgeführt werden. Diese sind, wie im vorigen Abschnitt erklärt, der integrierten Lösung zuzuordnen. Sie ersetzen die bisherigen normalen Datenpakete eines passiven Netzes. Die Ausführung geschieht in den Execution Environments (EEs) der Netzknoten, die auch als virtuelle Rechner bezeichnet werden.

3.1.1 Active IP Option

Als erste Möglichkeit wäre die *Active IP Option* zu nennen [tACM96]. Hier wird auf ein bereits vorhandenes für unterschiedliche Zwecke vorgesehenes Optionsfeld im Header der IP-Datagramme (IPv4/IPv6) zurückgegriffen. Man realisiert es in diesem Fall als „Active Option“, und es bietet dann die Möglichkeit hierin ein Programmfragment einzubetten. Internet-Netzknoten, die aktive Funktionen unterstützen, identifizieren die Active Option (anhand des Headers) und führen sie aus; herkömmliche passive Netzknoten ignorieren die Active Option und behandeln das aktive Paket als gewöhnliches IP-Datagramm. Zahlreiche verschiedene Codierungsmöglichkeiten stehen für die Programmfragmente zur Verfügung. Die Codierung kann binär oder als Quellcode einer höheren Programmiersprache erfolgen. Dabei ist zu unterscheiden, ob das Programm direkt Befehle erteilt oder im Netzknoten programmierte einfache Funktionen/Berechnungen (primitives) der Reihe nach aufruft. Demonstrationsbeispiele in [tACM96] zeigen anhand der Tcl-Codierung (Programmübertragung im Quellcode als Skript-Sprache) verschiedene praktische Anwendungen. Einfache Beispiele sind die Rückmeldung über die Ankunft der Datenpakete am Zielknoten oder die Angabe des zurückgelegten Pfades innerhalb des Netzes. Tcl gibt die Anweisungen, in welcher Reihenfolge Operationen durchgeführt werden müssen. Diese sind hier in C programmiert und Bestandteil des Tcl-Interpreters (Execution Environment) im Netzknoten.

Die Active IP Otion stellt eine relativ einfache Möglichkeit dar, ein aktives Netz zu gestalten, da sie auf bereits vorhandene Optionen des zugrundeliegenden IP-Protokolls aufbaut und deshalb nur die Funktionen innerhalb der Netzwerkschicht erweitert. Allerdings verfügt die Active IP Option nur über eingeschränkte Möglichkeiten hinsichtlich der Verwaltung und Bereitstellung von Ressourcen (Speicherplatz, Rechenleistung etc.) der Netzknoten, da hierfür in den zugrundeliegenden IP-Protokollen keine Optionen vorhanden sind.

3.1.2 Smart Packets

Im Gegensatz zur Active IP Option Architektur greift die *Smart Packets Architektur* (entwickelt von BBN Technologies) [SJSZ$^+$98] auf keine Optionen eines zugrundeliegenden Netzwerkdienstes (z.B. IP) zurück. Nur die Möglichkeit, dem Netzknoten durch einen Eintrag im

Header des Netzübertragungsrahmens mitzuteilen, ob es sich um ein Smart Packet handelt oder nicht, wird genutzt. Die Smart Packets Architektur besteht aus vier wesentlichen Komponenten: den Smart Packets als aktive Pakete, einer höheren Programmiersprache, einer EE sowie einer Sicherheitsarchitektur. Smart Packets werden zunächst in Übertragungsrahmen des Active Network Encapsulation Protocolls (ANEP) und dann in die Übertragunsrahmen des zugrundeliegenden Netzwerkprotokolls (z.B. IPv4 oder IPv6) eingebettet. Das ANEP Protokoll dient dazu, unterschiedliche Architekturen aktiver Netze kompatibel zu machen (entwickelt für das DARPA Projekt „Aktive Netze“). Den Aufbau eines kompletten Übertragunsrahmens am Beispiel des IP zeigt Abbildung 2. Die Nutzlast eines Smart Packets kann verschiedene Inhalte haben, die in den Steuerinformationen (im Typ-Feld) des Smart Packets identifiziert werden: Programmcode, Daten, Meldungen und Fehlermeldungen. Der Programmcode wird in den Netzknoten ausgeführt, Daten liefern Ergebnisse der ausgeführten Programme an die Herkunftsadresse zurück und Meldungen bzw. Fehlermeldungen dienen der Ablaufsteuerung.

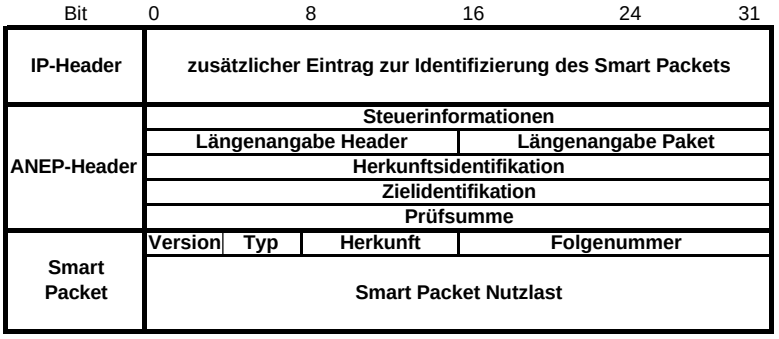

Abbildung 2: Eingekapseltes Smart Packet (vereinfachte Darstellung, angelehnt an [SJSZ+98])

Smart Packets werden durch benutzerspezifische Programme generiert. Ein sogenannter ANEP Daemon stellt den wesentlichen Bestandteil der Smart Packets EE dar und muß auch auf dem Quellrechner der Anwendung installiert sein. Er ist dafür zuständig, die Smart Packets in das Netz zu übergeben (d.h. die oben beschriebene Einkapselung der Smart Packets vorzunehmen) und die notwendigen Programmschritte der Smart Packets in den Netzknoten auszuführen. Herkömmliche Programmiersprachen (z.B. C) können zur Programmierung von Smart Packets nicht genutzt werden, da die Länge der Programmcodes in Smart Packets auf 1 Kbyte begrenzt ist. Eine geeignete höhere Programmiersprache für Smart Packets ist Sprocket. Sprocket [SJSZ+98] basiert auf Grundelementen von C, verfügt aber nur über wesentliche Befehle, die der Netzwerksteuerung- und überwachung dienen. Programme, die in Sprocket implementiert sind, werden in die Assemblersprache Spanner umgewandelt oder können direkt in Spanner implementiert werden. Smart Packets werden von der EE eines Netzknoten identifiziert und in Programmpakete umgewandelt, die den eigentlichen maschinenlesbaren Binärcode enthalten. Grundsätzlich können Smart Packets auch unerwünschte Funktionen innerhalb eines Netzes auslösen und sind deshalb ein Sicherheitsrisiko. Um die Sicherheit eines Smart Packet Netzes zu gewährleisten, prüft die EE eines Netzknotens die Integrität und die Herkunft eines Smart Packet Programms. Da die Rechenleistung eines Netzknoten beschränkt ist, muß ebenfalls die Komplexität der Programme überprüft werden, um eine Überlastung zu vermeiden. Die hierfür erforderlichen Informationen befinden sich in den Paketköpfen des ANEP und der Smart Packets (siehe Abbildung 2). Diese Sicherheitsfunktionen bietet die zuvor beschriebene Active IP Option nicht.

3.1.3 M0

Die *M0(M-zero)-Umgebung* (entwickelt an den Universitäten Berkeley (Kalifornien) und Zürich (Schweiz)) für sogenannte Messengers stellt eine Architektur dar, die sich deutlich von der bisher gebräuchlichen Definition computergestützter Kommunikation absetzt [Tsch97]. Der Schwerpunkt soll hier nicht mehr auf Datenübertragung sondern auf reiner Programmübertragung liegen. Mobile Programme greifen auf im Netz verteilte Datenspeicher zurück, die sowohl zur Speicherung von Programmcode als auch von Nutzdaten verwendet werden. Die Messengers übermitteln in dieser Architektur mobile (benutzerspezifische) Programmfragmente. Der Aufbau von Messengers ist sehr einfach: Sie bestehen aus einem Header, dem eigentlichen M0-Programmfragment und einem optionalen Datenfeld. Die Ausführung der M0-Programme geschieht in der M0-Umgebung (EE) der Netzknoten. Den logischen Aufbau einer M0-Umgebung zeigt Abbildung 3. Mehrere M0-Umgebungen (Netzknoten) müssen lediglich durch einen einfachen (unzuverlässigen) Datagramm-Dienst verbunden sein.

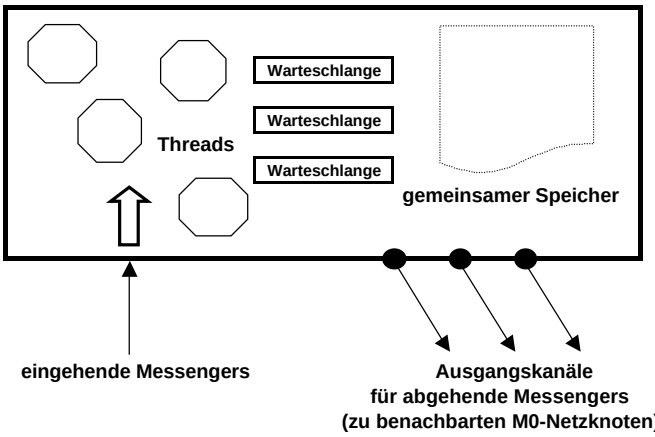

Abbildung 3: M0 Execution Environment

Messengers (Überbringer von Anweisungen), die einen Netzknoten erreichen, werden zunächst vom M0-Interpreter in Programm- bzw. Steuerpakete umgewandelt und zu sogenannten „Threads“ (Zusammenhänge, „rote Fäden“), die dann anonym sind und nicht miteinander kommunizieren können, zusammengefaßt. Die Threads reihen sich dann nach zufällig zugeordneten Schlüsseln, die sich im Header der Messengers befinden, in ein Warteschlangensystem ein, um auf die Ressourcen des Netzknotens zugreifen zu können. Gemeinsame Speicherplätze in den M0-Netzknoten können von den Threads zum Datenaustausch genutzt und eindeutig adressiert werden. Die Threads sind wiederum ihrerseits in der Lage, neue Messengers zu generieren, die sie auf den Ausgangskanälen an benachbarte M0-Netzknoten weiterleiten können, um dort bei Bedarf weitere Programmschritte ausführen zu lassen. Ressourcen der Netzknoten (Speicherplatz, Prozessorleistung etc.) werden den Threads mittels Accounts zugeteilt, von denen „Guthaben“ abgebucht wird. Ohne Guthaben besteht keine Möglichkeit zum Zugriff auf Ressourcen, und der Thread wird verworfen. Ein „Startguthaben“ wird jedem Messenger, der in einen M0-Netzknoten eintrifft und in ein Thread umgewandelt wird, zugeteilt. Dabei besteht vorher die Möglichkeit zu prüfen, wie „kostspielig“ der Zugriff auf die Ressourcen eines Netzknoten ist. Die „Kosten“, die in einem M0-Netzknoten entstehen, hängen besipielsweise von der aktuellen Last ab. Werden die „Kosten“ als zu hoch eingeschätzt, wird ein neuer Messenger mit dieser Mitteilung an die Herkunftsadresse zurückgeschickt, damit von dort wiederum neue Messengers an andere Netzknoten gesendet werden können. Praktisch bedeutet das folgendes: Messengers (bzw. die daraus entwickelten Threads) arbeiten sich „selbständig“ durch das Netz hindurch, um an verschiedenen Stellen Teilaufgaben durchzuführen, die wiederum Bestandteil einer übergeordneten Aufgabe (dem durch die Threads

gebildeten Zusammenhang) sind. Der in den Messengers übermittelte Programmcode ist in der sogenannten M0-Sprache geschrieben. Der M0-Interpreter ist in C implementiert.

Durch die Möglichkeit, innerhalb des gemeinsamen Speichers eines Netzknotens eindeutige Speicherstellen zu adressieren, besteht auch die Möglichkeit, Programmfragmente komplexer und damit in der Regel längerer Programme zwischenzuspeichern, wieder zusammenszusetzen und dann auszuführen. Diese Möglichkeit bieten die beiden vorher beschriebenen Architekturen nicht. Besondere Sicherheitsmechanismen bietet hingegen die M0-Architektur nicht. Sie geht davon aus, daß Zugriffsrechte auf Ressourcen der Netzknoten durch Messengers generell erlaubt sind und lediglich durch die Accounts der Threads geregelt werden (offene Sicherheitsarchitektur). Regelungen über generelle Zugriffsrechte auf Plattformen (Netzknoten) müssen vorher durch Plattformbetreiber und Benutzer ausgehandelt werden.

3.1.4 ANTS

Ein Netzwerk als verteilte Programmierumgebung zu betrachten, ist wesentlicher Bestandteil von *ANTS* (Active Node Transfer System) [WeGT98], einer Entwicklung des MIT. Wichtigste Komponenten dieser Architektur sind die zu übertragenden Pakete, die hier im wesentlichen eine Steuerfunktion übernehmen und auch hier als Kapseln („capsules“) bezeichnet werden, aktive Netzknoten, die Befehle der Kapseln ausführen sowie ein Mechanismus zur Verteilung von Programmcodes im Netz (code distribution mechanism). Die Kapseln können wie bei DAN (siehe Abschnitt 3.2.1 verschiedene identifizierbare Programme aufrufen, die die Nutzlast der Kapseln bearbeiten oder die Wegefindung im Netz beeinflussen können. Programme, die auf einem aktiven Knoten nicht vorhanden sind, werden durch den Code-Verteilungs-Mechanismus (code distribution mechanism) von benachbarten aktiven Knoten geladen. Dabei handelt es sich in der Regel um benutzerspezifische Programme. Während der Such- und Ladevorgänge der Programme werden die Kapseln in einen Ruhezustand (sleep) versetzt und bei erfolgreicher Programmübertragung wieder aktiviert. Kann eine Anfrage nicht befriedigt werden, wird die entsprechende Kapsel verworfen (Schonung der Ressourcen). Verwandt bedeutet, daß sie gemeinsame Daten und Programme benutzen. Verwandte Kapseln werden zu Code-Gruppen zusammengefaßt. Code-Gruppen werden vom Code-Verteilungs-Mechanismus mit zusammengehörenden Programmeinheiten versorgt. Als Protokoll bezeichnet man schließlich verwandte Code-Gruppen. Protokolle sind die Programme der Anwendungen bzw. der Benutzer, die an die ANTS Execution Environments übergeben und mittels der oben beschriebenen Hierarchie ausgeführt werden. Sie werden in JAVA programmiert und können in den Netzknoten nicht verändert werden. Diese Hierarchie zeigt Abbildung 4.

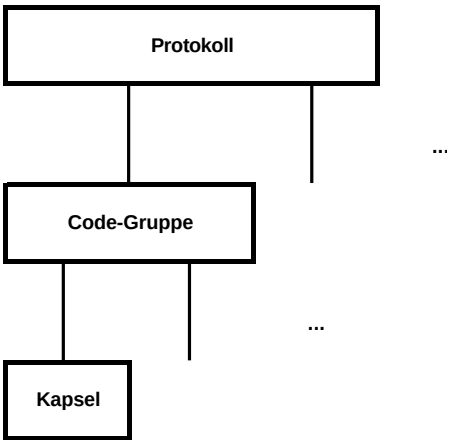

Abbildung 4: ANTS-Hierarchie (angelehnt an [WeGT98])

Abbildung 5: Format einer ANTS-Kapsel

Das erste Feld einer Kapsel gibt die Zugehörigkeit zu einem bestimmten Protokoll und den Kapsel-Typ an. Der Kapsel-Typ ordnet der Kapsel ihre spezifische Aufgabe innerhalb des Protokolls zu. Die wesentlichen Inhalte des gemeinsamen Paketkopfes sind Ziel- und Herkunftsadresse einer Kapsel. Im typabhängigen Paketkopf befinden sich spezifische Informationen, die auf dem Weg einer Kapsel durch das Netz ständig von den Netzknoten aktualisiert werden. Nutzdaten befinden sich im letzten Feld der Kapsel. Jede Kapsel besitzt im typabhängigen Paketkopf ein TTL-Feld (Time To Live), welches die noch verbleibende Lebensdauer der Kapsel angibt. Dieser Wert wird bei der Inanspruchnahme von Ressourcen reduziert. Bei TTL = 0 wird die Kapsel verworfen. Das TTL-Feld begrenzt also die angebotenen Ressourcen und soll so verhindern, daß die Netzknoten überlastet werden. Den schematischen Aufbau zeigt Abbildung 5.

In den ANTS-EEs sind zusätzlich einfache Routinen (node primitives) aufrufbar, wie z.B. Steuer- oder Speicherroutinen. Kapseln können somit beispielsweise Daten kurzzeitig in aktiven Knoten zwischenspeichern, neue Kapseln generieren oder Routing-Informationen gewinnen. Hiermit können Programme flexibel im Netz verteilt, erzeugt oder ausgeführt werden.

Zusammenfassend ausgedrückt stellt ANTS eine EE dar, mit deren Hilfe der Benutzer das Netzwerk als eine zusammenhängende Programmier- und Arbeitsumgebung verwenden kann.

3.2 Diskrete Architekturen

Übertragungspakete der Rahmenarchitektur aktiver Netzknoten enthalten keine vollständigen ausführbaren Programmcodes. Vielmehr tragen diese Datenpakete nur Verweise (Pointer) auf Programme, die entweder in den Netzknoten vorhanden sind oder von sogenannten Code-Servern (diskrete Lösung) oder anderen Quellen (integrierte Lösung, wenn die Programme direkt vom Benutzer stammen) geladen werden müssen. Jedes Programm, das auf einem aktiven Netzknoten verfügbar ist, kann eindeutig identifiziert werden. Identifikationseinträge im Header eintreffender Datenpakete rufen die gewünschte Funktion auf und übergeben dabei evtl. notwendige Parameter. Diese Lösung bietet ein hohes Maß an Sicherheit, da die Verwaltung der ausführbaren Programme beispielsweise von einer zentralen Netzwerksteuerung (wenn man sich für die diskrete Lösung, die in diesem Abschnitt behandelt wird, entschieden hat) durchgeführt werden kann. Die Flexibilität hinsichtlich der Aktualität und der Vielseitigkeit der Programme läßt bei dieser ziemlich statischen Lösung jedoch zu wünschen übrig. Deshalb findet man in der Praxis Abwandlungen.

3.2.1 DAN

Die *DAN-Architektur* (Distributed Code Caching for Active Networks) bietet die Möglichkeit auf Code-Server zuzugreifen [Psou99]. Code-Server verfügen über ein breites Angebot an Programmen für unterschiedliche Plattformen von verschiedenen Entwicklern. Sie sind in den Knoten eines aktiven Netzes verzeichnet. Wird nach dem oben beschriebenen Schema ein Programm aufgerufen, das auf einem Netzknoten nicht vorhanden ist, kann es von einem Code-Server heruntergeladen werden und bleibt dann auch für spätere Aufrufe lokal gespeichert, um erneute unnötige Ladevorgänge, die das Netz belasten, zu vermeiden. Die neuesten Programmentwicklungen werden zentral auf dem Code-Server verwaltet und stehen dann dem

gesamten Netz bei Bedarf zur Verfügung. Bei dieser Architektur steht also das Ziel im Vordergrund, eine möglichst aktuelle und vielseitige Programmauswahl zur Verfügung zu stellen. Die Verwaltung des Code-Servers bestimmt bei dieser Architektur die Leistungsfähigkeit des Netzes.

3.2.2 CANEs

CANEs (Composable Active Network Elements) ist ebenfalls ein von der DARPA unterstütztes Projekt für aktive Netze. CANEs ist eine Excution Environment, die unter gewöhnlichen Betriebssystemen (NodeOS) installiert werden kann. CANEs stellt eine Umgebung mit grundlegenden Funktionen (underlying programms) zur Verfügung. Benutzerspezifische Programme, die kontrolliert in das Netz eingespielt werden, ergänzen dann die Grundfunktionen. In [tAnn99] wird CANEs in Verbindung mit einem besonderen Betriebssystem, *Bowman*, vorgestellt. Bowman unterstützt mehrere EEs auf einem Netzknoten gleichzeitig und setzt auf ein vorhandenes „normales“ Betriebssystem auf. Wesentliche Komponenten von Bowman sind: „Channels“, „A-flows“, „State Store“ sowie ein Mechanismus zur Integration ergänzender Module. Channels sind die Kommunikationswege von Bowman. Ihre Besonderheit ist, daß sie auf die zu übertragenden Daten zugreifen und sie verändern können. Mehrere Channels können zu einem Link zusammengefaßt werden und somit einen abstrakten Netzknoten (virtuelles Netz) bilden. A-flows führen Programme unterschiedlichster Art aus. State-Stores speichern Zustände von A-flows. Man könnte sie auch als Systemregistrierung bezeichnen. Eingehende Datenpakete werden identifiziert und an entsprechende Channels und/oder A-flows weitergeleitet. Datenpakete, die kein A-flow durchlaufen müssen, werden direkt an einen Ausgangs-Channel weitergeleitet (cut-through). Bowman stellt der EE die Ressourcen eines Netzknotens in funktionsspezifischer Form zur Verfügung. Die EE kann nicht direkt auf Ressourcen der Netzknoten zugreifen.

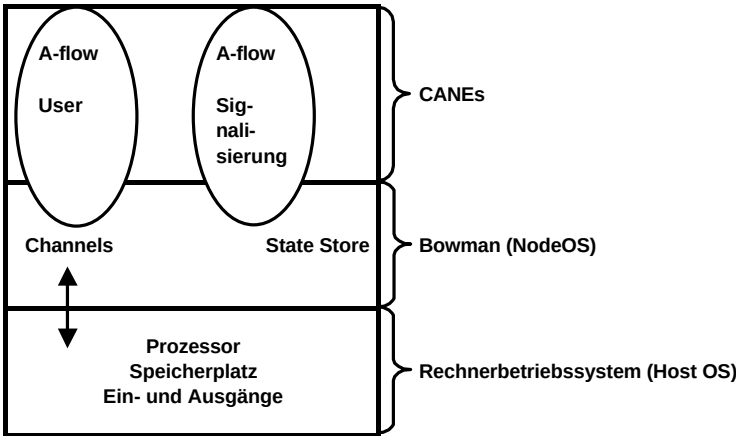

Abbildung 6: Schematischer Aufbau eines aktiven Knotens mit Bowman und CANEs

Die CANEs EE bietet nun, hier in Verbindung mit Bowman [PoIE01], zwei Programmarten. Ein allgemeines Programm (underlying program), das einheitlich auf alle durchgeleiteten Pakete angewendet wird sowie ein benutzerspezifisches Programm (injected program). Benutzerspezifische Programme werden bei Bedarf in die allgemeinen Programme in sogenannte „slots“ integriert. Slots sind markierte „Leerstellen“ innerhalb des allgemeinen Programms, die überschrieben werden können. Die allgemeinen Prgramme sind beispielweise für die Wegefindung (Routing) zuständig. Benutzerspezifische Programme sind entweder auf dem Netzknoten bereits vorhanden oder können bei Bedarf durch die EE von außen (z.B. benachbarten Knoten) geladen werden.

Abbildung 6 zeigt schematisch die Zusammenhänge zwischen CANEs und den zugrundeliegenden Betriebssystemen. Mehrere A-flows können im System existieren. In Abbildung 6 sind beispielhaft ein User A-flow gezeigt, daß die Programme (also das allgemeine und ein oder mehrere benutzerspezifische Programme) von CANEs ausführt sowie ein A-flow, daß Steuerfunktionen trägt. Diese werden über die Channels an die erforderlichen Adressen weitergeleitet, diese können physikalisch oder virtuell sein.

3.3 Kombination integrierter und diskreter Architekturen

Die in diesem Kapitel vorgestellten Architekturen kombinieren die beiden grundlegenden Rahmenarchitekturen. Aktuelle und kürzere Programme werden von aktiven Paketen übertragen (integrierte Lösung), wobei sich komplexe und längere Prgramme in den Netzknoten befinden (integriert oder diskret geladen). Der Benutzer kann nun entscheiden, welche Kombination die richtige für die jeweilige Anwendung ist.

3.3.1 SwitchWare

Eine Möglichkeit hierfür stellt die *SwitchWare Architektur* der Universität Pennsylvania dar [AAHK+98]. In der SwitchWare Architektur sind drei Schichten definiert: aktive Pakete (mobil), aktive Programme (immobil) und die Infrastruktur der Netzknoten (siehe Abbildung 7). Die aktiven Pakete entsprechen der integrierten Rahmenarchitektur und die aktiven Programme (in aktiven Netzknoten) der diskreten Rahmenarchitektur.

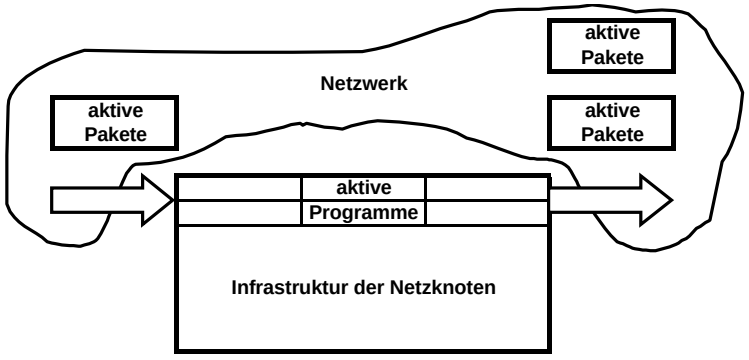

Abbildung 7: SwitchWare Architektur

Aktive Pakete transportieren einfache Programme mit minimaler Funktionalität. Sie werden hier in PLAN (Programming Language for Active Networks) implementiert und sind in der Lage, andere PLAN-Programme auf entfernten Netzknoten auszuwerten. Der hierfür zuständige Mechanismus ermöglicht es den aktiven Paketen, über das gesamte Netz hinweg zu agieren. PLAN kann nur minimale Funktionen mit stark beschränktem Ressourcenzugriff ausführen, weshalb auf dieser Schicht keine besonderen Sicherheitsmechanismen erforderlich sind. Die aktiven Programme innerhalb der Netzknoten (auch Switchlets genannt), die von den aktiven Paketen bei Bedarf aufgerufen werden, sind wesentlich leistungsfähiger als die mobilen Programme und können in herkömmlichen Programmiersprachen implementiert werden. Dabei sind sie in der Lage, in großem Umfang auf die Ressourcen des Netzknotens zuzugreifen. Sie sind immobil, weil sie nur mit Hilfe der aktiven Pakete mit anderen Netzknoten kommunizieren können. Die unterste Schicht der SwitchWare Architektur stellt die Infrastruktur der Netzknoten dar. Sie ist, verglichen mit den beiden oberen Schichten, unflexibel, dafür jedoch für die Sicherheit der aktiven Netzknoten zuständig. SANE (Secure Active Network Environment) stellt die notwendigen Funktionen zur Verfügung, die die Infrastruktur (Prozessor, Speicher etc.) unabhängig von den aktuellen Anwendungen vor Überlastung oder

Beschädigung schützt. Dabei werden nur die Switchlets überprüft. Zusammenfassend kann man SwitchWare so beschreiben: Konträre Anforderungen wie Sicherheit und hohe Flexibilität werden auf unterschiedliche Schichten mit jeweils speziellem Aufgabengebiet verteilt, um so insgesamt ein ausgeglichenes Ergebnis zu erzielen.

3.3.2 NetScript

Die *NetScript Architektur* der Universität Columbia [TeWe96] soll die dynamische Programmierung eines Netzes ermöglichen. Das Netz wird in der NetScript Architektur als virtuelles NetScript Netzwerk (NetScript virtual network, NVN) bezeichnet. Es besteht aus virtuellen Netzwerkrechnern (virtual network engines, VNEs), die über virtuelle Links (virtual links, VLs) miteinander verbunden sind. Die virtuellen Links und die virtuellen Netzwerkrechner müssen nicht mit den physikalischen (also den tatsächlichen Netzknoten) übereinstimmen. Mehrere VLs und VNEs können sich auf einem physikalischen Netzknoten befinden. VLs können auch mehrer NVNs miteinander verbinden.

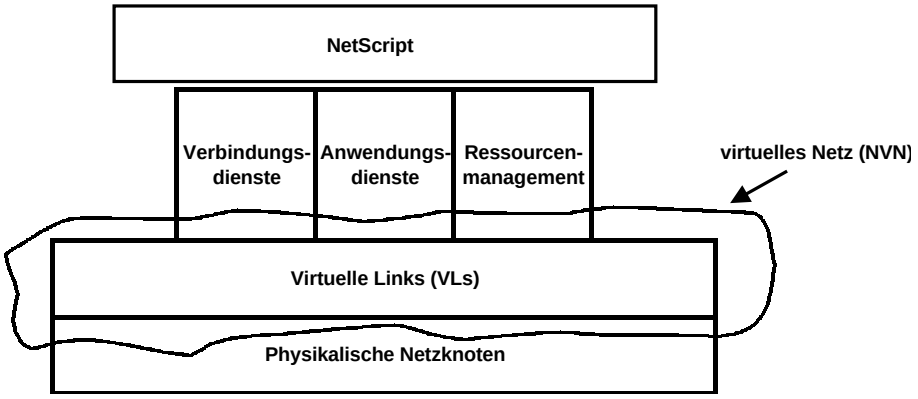

Abbildung 8: NetScript Architektur

Das NVN bildet die Basis der NetScript Architektur. Hierauf bauen verschiedene weitere Teilschichten sowie die auszuführende Programme auf. Die Teilschicht der Verbindungsdienste koordiniert das Zusammenwirken virtueller und physikalischer Links bzw. Rechner. Die Ececution Environment dieser Architektur stellt die Teilschicht Anwendungsdienste dar. Hier werden die Befehle der Programme (Net Script Programme in den Paketen und beliebige Programme, die in den Netzknoten gespeichert werden) ausgeführt. Das Ressourcen Management Modul steuert die Ressourcen des physikalischen Netzknotens. Abbildung 8 zeigt den Aufbau der NetScript Architektur.

NetScript Programme, die in den aktiven Paketen übertragen werden, steuern nun die Datenströme innerhalb des Netzes und konfigurieren virtuelle Netzabschnitte. Hierbei können innerhalb des NVNs viele Transaktionen parallel zueinander ablaufen, weshalb eine Synchronisation erforderlich ist. Datenflußsteuerprograme (wie NetScript) kommunizieren deshalb über klar defininierte Schnittstellen mit der lokalen Hard- und Software und erfordern keine speziellen plattformabhängigen Synchronisationsmechnismen. Dies ist ein Vorteil der NetScript Sprache, weil sie innerhalb eines NVN auf zahlreichen unterschiedliche Plattformen funktionieren muß. Die Datenströme (Datenpakete) sind in der NetScript Architektur Nachrichten (Meldungen) sowie die In- und Outputs verschiedener im virtuellen Netz verteilter Programme, die in beliebigen Programmiersprachen (z.B. C) implementiert sein können. Auf diese Weise entsteht eine verteilte Programmierumgebung, die dem Benutzer als (virtuelle) Einheit zur Verfügung steht.

3.4 Zusammenfassender Vergleich

Die Ausführungen der vorangehenden Abschnitte zeigen, daß jede Architektur ihre besondere Stärken hat. Bei den integrierten Lösungen ist eine Tendenz zu besonderer Flexibilität erkennbar, die, wenn man einen hohen technischen Aufwand in Kauf nimmt, mit guter Leistungsfähigkeit ergänzt wird. Bei Active IP-Option und bei SmartPackets wird zur Zeit hauptsächlich mit Netzwerksteuerungs- und Netzwerküberwachungsfunktionen experimentiert, weshalb hier nur mittlere Flexibilität angegeben werden kann. Andere Anwendungen sind aber auch hier denkbar. Die besondere Stärke der diskreten Architekturen ist die Sicherheit durch die Kontrollmöglichkeit der Programmherkunft. Kombinierte Architekturen erzielen brauchbare Kompromisse und können als Universallösungen angesehen werden. Durch die Möglichkeit, eine virtuelle Netztopologie aufzubauen, bietet NetScript eine höhere Flexibilität als SwitchWare, das sich wiederum durch die Schichteneinteilung bei allen Anforderungen keine gravierende Schwächen erlaubt. Eine perfekte Architektur, die allen Anforderungen gleichermaßen gerecht wird, gibt es momentan nicht. Dies läßt zunächst den Schluß zu, daß man je nach Anwendungsgebiet zwischen gebotenem Leistungsumfang und erforderlichem Aufwand abwägen muß. Allerding wird hierbei der grundsätzlichen Forderung nach Flexibilität, also der Unabhängigkeit der Netzarchitektur von bestimmten Anwendungen, nicht Rechnung getragen. Aus rein technischer Sicht müßte man also integrierte Lösungen bevorzugen, wobei hier ANTS und M0 bei gleichzeitig hoher Leistungsfähigkeit geeignet erscheinen. Eine zusammenfassende Übersicht der wesentlichen Eigenschaften zeigt Abbildung 9.

	Flexiblität	Leistungsfähigkeit	Sicherheit	Aufwand
Active IP Option	]	]	–	–
Smart Packets	]	]	]	–
M0	+	+	]	+
ANTS	+	+	]	+
DAN	–	+	+	–
CANEs	]	+	+	]
SwitchWare	]	]	]	]
NetScript	+	]	]	]

+ hoch,] mittel, - gering

Abbildung 9: Vergleichsübersicht

4 Schlussfolgerungen und Aussicht

Die vorgestellten Architekturen zeigen, daß aktive Netze durch ihre zahlreichen Anpassungsmöglichkeiten der schnellen Weiterentwicklung von Anwendungssoftware und Endsystemen eher folgen können als herkömmliche passive Netze. Die meisten Lösungen bieten die Voraussetzung, daß die Programmierung von Netzen nicht mehr abhängig sein wird von der Entwicklung der im Netz verwendeten Hard- und Software. Langwierige Normungsverfahren, die notwendig sind, um aufeinander abgestimmte Hard- und Softwareentwicklungen mit anderen Systemen kompatibel zu machen und damit die Weiterentwicklung der Technologien bremsen können, sollen so vermieden werden. Die Weiterentwiclung der Hardware muß sich in erster Linie bei aktiven Netzen darauf ausrichten, die allgemeine Leistungsfähigkeit von Netzknoten (Prozessorleistung, Speicherplatz etc.) zu verbessern, da diese bei aktiven Lösungen durch die Verlagerung von bisherigen Endgerätefunktionen in das Netz wesentlich stärker beansprucht werden.

Literatur

[AAHK+98] D. Scott Alexander, William A. Arbaugh, Michale W. Hicks, Pankaj Kakkar, Angelo D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles und Jonathan M. Smith. The Switch Ware Active Network Architecture. Technischer Bericht, University of Pennsylvania, Juli 1998.

[Calv99] Ken Calvert. Architectural Framework for Active Networks, Version 1.0. Technischer Bericht, Department of Computer Science, University of Kentucky, Juli 1999.

[PoIE01] Alaska Proceedings of IEEE OpenArch 2001, Ancharage (Hrsg.). *Active Reliable Multicast on CANEs: A Case Study*, April 2001.

[Psou99] Konstantinos Psounis. Active Networks: Applications, Security, Safety and Architectures. *IEEE Communications Surveys*, 1999.

[SJSZ+98] Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell und Craig Partridge. Smart Packets for Active Networks. Technischer Bericht, BBN Technologies, Januar 1998.

[tACM96] Ireland 7th ACM SIGOPS European Workshop, Connemara (Hrsg.). *The Active IP Option*, September 1996.

[tAnn99] IL 37th Annual Allerton Conference, Monticelle (Hrsg.). *Bowman and CANEs: Implementation of an Active Network*, September 1999.

[TeWe96] David L. Tennenhouse und David J. Wetherall. Towards an Active Network Architecture. *Computer Communications Review* 26(2), 1996.

[Tsch97] Christian F. Tschudin. The Messenger Environment MO - a Condensed Description. Technischer Bericht, The Teleinformatics and Operating Systems Group, University of Geneva, Mai 1997.

[WeGT98] David J. Wetherall, John V. Guttag und David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. Technischer Bericht, Software Devices and Systems Group Laboratory for Computer Science, Massachusetts Institute of Technology, April 1998.

Abbildungsverzeichnis

Ressourcenverwaltung in aktiven Netzen

Marco Breiter

Kurzfassung

Aktive Netzwerke erweitern die Funktionalität traditioneller Netze, in denen Daten lediglich von Netzknoten transparent weitergeleitet werden, durch die Möglichkeit, eine auf einzelne Pakete abgestimmte Verarbeitung in den Netzknoten selbst vorzunehmen. Zusätzlich können die Benutzer die Funktionsweise des Netzes aktiv beeinflussen, indem sie Programme auf Netzknoten einbringen. Da diese Programme auf sehr heterogenen Netzknoten ausgeführt werden sollen, stellt sich die Frage, wie man die Verwaltung und den Zugriff auf Ressourcen gewähren kann, um einerseits ein hohes Maß an Flexibilität zu erreichen und andererseits Sicherheit und Effizienz sicherzustellen. Die NodeOS Schnittstelle, die beispielsweise in Janos implementiert worden ist, stellt einen Ansatz dazu dar, der im folgenden vorgestellt wird.

1 Einleitung

Die rasante Entwicklung des Internet Mitte der 90er Jahre brachte einen vollkommen neuen Grad an Vernetzung mit sich. Eine stark wachsende Zahl von immer leistungsfähiger werdenden Einzelrechnern ist mittlerweile miteinander gekoppelt. Galt in der Anfangszeit der auf TCP/IP basierenden Netze noch das Paradigma, Daten möglichst schnell und weitgehend transparent von einem Netzknoten zu anderen weiterzuleiten, haben in der Zwischenzeit weitere Ansätze an Bedeutung gewonnen. Bei genauerer Betrachtung der heutigen Rechnernetze fällt auf, dass Pakete an sehr vielen Stellen nicht mehr einfach nur weitergereicht werden, sondern oft sehr spezielle Verarbeitungsschritte durchgeführt werden, um Sicherheit, Effizienz oder eine spezielle Dienstgüte zu gewährleisten.

1.1 Beispiele für Verarbeitung in Netzwerken

Netzknoten werden häufig an Übergängen von Netzwerken als Firewalls genutzt, um die Sicherheit lokaler Netze gegenüber außenstehenden zu wahren. Sie filtern die einkommende Datenströme, und lassen nach spezifizierten Regeln erwünschte Pakete transparent passieren, während andere blockiert werden. An neue Protokolle und Applikationen muss eine Firewall speziell angepasst werden, falls dies überhaupt möglich ist. Eine Automatisierung dieses Prozesses wäre von großem Vorteil.

HTTP-Proxy-Server bieten einen für den Nutzer transparenten Dienst an. Um Netzlast zu minimieren werden häufig aufgerufene Inhalte zwischengespeichert. Üblicherweise befinden sich diese Server an den Übergängen von lokalen zu öffentlichen Netzen. Eine übergreifende Kooperation zwischen Proxy-Servern, die sich hierarchisch gesehen benachbart in unterschiedlichen Verantwortungsbereichen des Netzes befinden, ist aber nicht vorgesehen. Da zudem ein großer Teil heutiger Internetseiten dynamisch erzeugt wird, können Proxy-Server bei weitem nicht das Potential erreichen, das theoretisch möglich wäre um Übertragungszeiten zu minimieren und Bandbreite einzusparen.

Durch die immer leistungsfähiger werdenden Computer privater Nutzer und die Konvergenz verschiedener Medien gewinnen multimediale Anwendungen eine immer wichtiger werdende Rolle. Verteilte Multimedia-Anwendungen nehmen große Übertragungsbandbreiten in Anspruch. Gleichzeitig werden sie von vielen Netzteilnehmern genutzt und stellen damit eine große Belastung für das Gesamtnetz dar. Die Standardisierung eines speziellen Übertragungsprotokolls für multimediale Daten ist ein sehr langwieriger Prozess und dauert Jahre, trotzdem ist das keine Garantie dafür, dass sich ein solches Protokoll wie z.B. Multicast in der Praxis durchsetzt. Um multimediale Inhalte dennoch effizient übertragen zu können, bleibt als Alternative oft lediglich eine Kooperation zwischen Inhaltsanbietern und Netzbetreibern. An strategisch günstigen Netzknoten werden Inhalte hierarchisch zwischengespeichert, um die gesamte Netzlast und die Übertragungsdauer einzelner Dateien zu minimieren. Ein solches Vorgehen setzt natürlich eine entsprechende Marktposition voraus, ohne die eine Durchsetzung unmöglich wäre. Dabei bleibt aber zu beachten, dass hierbei immer nur eine einzelne Anwendung bzw. Protokoll unterstützt wird, und andere Inhaltsanbieter im allgemeinen trotz Verwendung desselben Protokolls nicht von bestehenden Lösungen profitieren können.

1.2 Probleme heutiger Netze

Alle diese Ansätze sind sehr speziell auf die verfügbare Hardware und Netztopologie abgestimmt, und wie viele ähnlich geartete Fälle im wesentlichen nur als ad hoc Lösungen anzusehen. Das Grundproblem der heutigen großen Netzwerke insbesondere des Internet besteht in der geringen Anpassungsfähigkeit an neue Anforderungen. Die Entwicklung und der Einsatz neuer Protokolle, die effizientere Datenübertragung gestatten oder mit neuen Leistungsmerkmalen aufwarten können, gestaltet sich sehr schwierig. Es dauert außerdem sehr lange, bis eine Anpassung an neue Technologien stattfindet.

Ein Lösungsansatz, der die Flexibilität besitzt, diese Probleme auf einheitliche Weise zu begegnen und gleichzeitig Potential für eine vollkommen neue Netzwerkanwendungen hat, sind aktive Netze. Leider gibt es bislang noch verhältnismäßig wenige wissenschaftliche Arbeiten, die sich mit der Ressourcenverwaltung beschäftigen. Im folgenden werden deshalb das NodeOS Referenzmodell und ein darauf aufbauendes Betriebssystem für aktive Knoten namens Janos vorgestellt, die sich der Verwaltung und Kontrolle von Hardwareressourcen aktiver Knoten widmen.

2 Aktive Netze

2.1 Definition

Ein aktives Netzwerk ist ein Rechnernetz, in dem Netzknoten eine auf einzelne Pakete abgestimmte Verarbeitung durchführen. Die Nutzer selbst können das Verhalten des Netzwerks beeinflussen, indem sie sogenannte „aktive Programme“ (engl. active applications, AA) in das aktive Netz einbringen. In einem aktiven Netz sind in den einzelnen Paketen Programme eingebettet oder ihnen zugeordnet, welche dann auf jedem passierten Netzknoten ausgeführt werden. Diese aktiven Programme erfüllen jedoch nicht nur die Aufgabe des klassischen Routings. Der Inhalt der Pakete selbst kann durch sie angepasst werden, um speziellen Anforderungen gerecht zu werden. Pakete transportieren also immer noch Daten zu ihrem Ziel, aber nun können sie das mit wesentlich größerer Flexibilität tun. Prinzipiell werden aktive Netzwerke in zwei unterschiedliche Ansätze eingeteilt, je nachdem ob Programme und Daten gemeinsam oder in getrennten Paketen übertragen werden.

2.2 Integrierter und diskreter Ansatz

Der diskrete Ansatz zeichnet sich dadurch aus, dass Daten und Programme separat übertragen werden. Man trennt in dieser Architektur die Verarbeitung der Pakete von dem Einbringen von Programmen auf aktiven Netzknoten. Statt der Bezeichnung aktives Netzwerk wird hier auch häufig von programmierbaren Netzen gesprochen. Für den Anwender ändert sich in dieser Architektur relativ wenig. Pakete werden weiterhin an programmierbare Netzknoten geschickt, ganz ähnlich wie das in heutigen Netzen der Fall ist. Beim Eintreffen eines Paketes wird dessen Paketkopf untersucht und abhängig vom Inhalt an das zugeordnete Programm weitergereicht, welches dann ausgeführt wird. Dieses Programm verarbeitet dann aktiv das einkommende Paket und verändert bei Bedarf auch dessen Inhalt. Dadurch lässt sich schon ein recht hoher Grad an Flexibilität erreichen, da abhängig vom Paketkopf und speziellen Protokolleinträgen entschieden werden kann, welches aktive Programm gestartet werden soll und so auf spezielle Anwendungs- oder Nutzerbedürfnisse eingegangen werden kann.

Dem gegenüber steht der integrierte Ansatz. Hier enthält jedes einzelne Paket ein ausführbares Programm. Man nennt ein solches Paket deshalb auch eine Kapsel (engl. capsule). Programme und Daten werden gemeinsam in einem Paket integriert und so verschickt. Erreicht eine Kapsel dann einen aktiven Knoten, wird der Inhalt an eine spezielle Ausführungsumgebung weitergeleitet, in der er sicher ausgeführt wird. Das aktive Programm kann so den Inhalt des eigenen Pakets manipulieren, mit bereitgestellten Funktionen auf externe Ressourcen außerhalb der Ausführungsumgebung zugreifen und Kapseln an andere Netzknoten verschicken.

Mit programmierbaren Netzen lassen sich auf verhältnismäßig einfache Weise Sicherheitsrisiken minimieren, wenn aktive Programme nur von einem verantwortlichen Administrator nach entsprechender Authentifizierung auf den Netzknoten eingebracht werden dürfen, um beispielsweise Konfigurationsänderungen vorzunehmen. Von Vorteil ist weiterhin, dass keine gravierende Umstellung zu bisherigen Paketen vorgenommen werden muss, da Netzknoten lediglich eine Schnittstelle für Pakete mit Programmen hinzugefügt werden muss. Zudem ist eine sehr effiziente Bearbeitung konventioneller Pakete sichergestellt, die schnell weitergeleitet werden. Andererseits büßt man damit auch viel der Flexibilität ein, die der integrierte Ansatz verspricht.

3 Referenzmodell NodeOS

3.1 Architekturansatz

Ein wesentliches Ziel aktiver Netze ist es, die Programmierbarkeit des Netzes so flexibel wie möglich zu gestalten und gleichzeitig die Ausführung von aktiven Programmen auf möglichst vielen Netzknoten zu gewährleisten. Die Unterstützung verschiedener Ausführungsumgebungen und Programmiersprachen erlaubt es zum Beispiel, auf besondere Sicherheitsbedürfnisse einzugehen und gleichzeitig verschiedene Ansätze der Programmierung zu nutzen. Das vom Defense Advanced Research Projects Agency (DARPA) finanzierte Active Network Forschungsprogramm hat deshalb in ein Referenzmodell erarbeitet [Grou01]. Um den bereits angesprochenen Grad an Flexibilität zu erreichen, und möglichst viele Programme auf unterschiedlichen Netzknoten nutzen zu können, wurde eine Unterteilung in mehrere Schichten vorgenommen. Das eigentliche Betriebssystem, hier NodeOS genannt, wird von der Ausführungsumgebung (engl. execution environment, Abk. EE) für aktive Programme getrennt. So kann ein einzelner Knoten einfacher mehrere Programmiersprachen unterstützen und eine größere Zahl aktiver Pakete ausführen. Andererseits wird so der Prozess stark vereinfacht, eine einzelne Ausführungsumgebung, welche eine bestimmte Sprache unterstützt, auf heterogene Netzknoten mit verschiedenen Hardwareressourcen zu portieren.

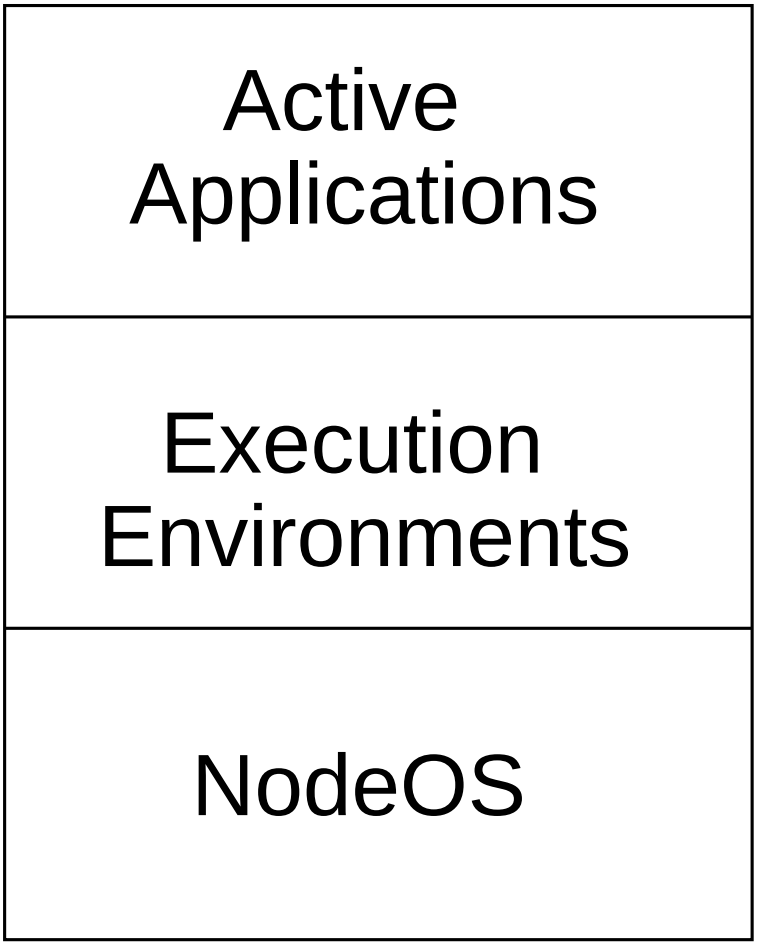

Abbildung 1: DARPA Referenzmodell für aktive Knoten

3.2 Betriebssystem

Mit NodeOS wurde eine einheitliche Schnittstelle definiert, um die Entwicklung der konkreten Betriebssysteme nicht zu sehr einzuschränken. Ein Betriebssystem muss Funktionen dieser Schnittstelle implementieren, um zu NodeOS kompatibel zu sein, kann sie aber zusätzlich um eigene Funktionen erweitern. Dadurch können Ausführungsumgebungen entwickelt werden, die unabhängig von der verwendeten Hardware auf verschiedensten aktiven Knoten einsetzbar sind. Das Betriebssystem hat also die Aufgabe, Hardwareressourcen des aktiven Knotens zu verwalten und den Zugriff auf sie zu kontrollieren. Details der konkreten Hardware sind so für die Ausführungsumgebungen nicht mehr relevant. Sie greifen lediglich über die fest definierte Schnittstelle NodeOS zu und nutzen unter anderem die verfügbaren Übertragungsbandbreiten, den Speicher und Verarbeitungskapazitäten. Das Betriebssystem kann so gleichzeitig mehrere Ausführungsumgebungen parallel ausführen und verteilt die Ressourcen entsprechend.

3.3 Ausführungsumgebung

Während die Betriebssysteme noch direkt auf der Hardware aufsetzen, und speziell auf den Netzknoten abgestimmt sein müssen, abstrahieren die Ausführungsumgebungen davon und führen aktive Applikationen aus. Den Ausführungsumgebungen obliegt es auch, ein entsprechendes Sicherheitsniveau zu etablieren. Sie stellen den Programmierern nur einen beschränkten Satz von Funktionen zur Verfügung, die das Betriebssystem selbst bietet. So muss ein Mittelweg zwischen einer entsprechend mächtigen Programmierschnittstelle auf der einen Seite und Authentifizierungsmechanismen und Sicherheitsaspekten auf der anderen Seite gefunden werden. Die Verantwortlichkeit für Ressourcenverwaltung und Kontrolle teilen sich Betriebssystem und die Ausführungsumgebungen auf ihren jeweiligen Ebenen, deshalb ist eine klare Trennung schwierig.

3.4 Aktive Programme

Ein aktives Programm führt Verarbeitungsschritte für einen bestimmten Paketfluss aus. Dabei kann das aktive Programm mit anderen Programmen kommunizieren und interagieren, auf Ressourcen des aktiven Knotens zugreifen, den Inhalt der zugeordneten Pakete modifizieren und die Pakete selbst an andere Knoten weiterschicken. Je nach verwendeter Ausführungsumgebung muss ein aktives Programm in einer speziellen Programmiersprache geschrieben sein.

Allgemein kann man verschiedene Klassen von Programmiersprachen für aktive Knoten unterscheiden, die abhängig vom verwendeten Ansatz individuelle Vor- und Nachteile haben. Einerseits könnte eine Ausführungsumgebung sehr schlank gehalten sein, und die Ausführung von binären Programmen in Maschinensprache gestatten. Dieser Ansatz ist zum Beispiel für einen programmierbare Knoten sinnvoll, wie er oben beschrieben wurde. Es lassen sich so spezielle Eigenschaften der Hardware ausnutzen, um die Programme stark zu optimieren und den Durchsatz von Paketen zu steigern. Auf der anderen Seite führt dies auch dazu, dass solche aktiven Programme kaum portabel sind. Zudem muss ein aktiver Knoten auf die Sicherheit und Funktionsfähigkeit der Programme implizit vertrauen können.

Hochsprachen und spezielle interpretierte Sprachen sind die Alternativen. Programme laufen dort wie bei Java in einer virtuellen Maschine ab oder werden wie bei Skriptsprachen interpretiert und sind so von der eigentlichen Hardware isoliert. Die Entwicklung aktiver Applikationen gestaltet sich mit Hochsprachen verhältnismäßig einfach und schnell. Dieselbe Effizienz wie Programme in Maschinensprache können diese interpretierten Applikationen allerdings selten erreichen. Trotzdem sind diese Programmiersprachen für den Einsatz in aktiven Knoten vielversprechend, da sich damit Sicherheit besser durchsetzen lässt und die Hardwareressourcen leichter kontrolliert und verwaltet werden können.

4 Ressourcenverwaltung

Aktive Netzwerke etablieren eine neue Art der Verarbeitung der Pakete im Netz. Dabei ist das Ziel allerdings nicht, vollkommen neue Netze zu etablieren, sondern auf der verfügbaren Infrastruktur aufzubauen. Die meisten der heutzutage verwendeten Rechnernetze wie das Ethernet basieren auf dem Prinzip des konkurrierenden Zugriffs. Damit geht aber auch eine wichtige Einschränkung einher: Dienstgüte kann nicht garantiert werden. Das lässt sich auch durch den Einsatz aktiver Knoten im Netzwerk nicht ändern. Um diesem Ziel trotzdem so gut wie möglich zu entsprechen ist es von besonderer Bedeutung, die Ressourcen, auf deren Nutzung Einfluss ausgeübt werden kann, richtig zu verteilen.

4.1 Vergleich zwischen aktiven und passiven Netzwerken

In passiven Netzen ist im wesentlichen nur die Übertragungsbandbreite von Belang. Auf aktiven Knoten werden Programme ausgeführt, wodurch sich eine erheblich größere Bedeutung der Ressourcenverwaltung im allgemeinen ergibt. Neben der Übertragungsbandbreite, die immer noch eine sehr große Rolle spielt, erhalten die aktiven Programme Rechenzeit und Speicher, um ihren Aufgaben gerecht zu werden. Diese Ressourcen müssen verwaltet und kontrolliert werden, um die Funktionsfähigkeit des Knotens insgesamt zu gewährleisten und jedes aktive Programm entsprechend seines Bedarfs zu bedienen. Wenn Dienstgüte garantiert werden soll, muss die Möglichkeit bestehen, Einschränkungen oder Kontingentierungen der Nutzung vorzunehmen. Neben der reinen Verwaltung der Ressourcen sind deshalb Wege zur Kommunikation zwischen den aktiven Programmen zur Verfügung zu stellen.

4.2 Einheitliche Ressourcenbeschreibung - NodeOS

Das Referenzmodell NodeOS sieht deshalb eine einheitliche Ressourcenbeschreibung vor, um von einer konkreten Hardware zu abstrahieren. Da bei einem aktiven Netzwerk trotz aller Verarbeitung immer noch ein Paketfluss im Vordergrund steht, wird zu dessen Kapselung das Konzept der Domäne verwendet.

4.2.1 Domänen

Um die Nutzung der verschiedenen Ressourcen einem Verursacher zuordnen zu können und Kontrollmechanismen gezielt zu verwenden, nutzt man Domänen (engl. domains). Domänen enthalten alle Ressourcen, die nötig sind um einen bestimmten Paketfluss (packetflow) zu beschreiben. Typischerweise sind das zumindest Ein- und Ausgabekanäle um die Daten zu empfangen oder zu senden, Speicherbereiche um Daten zu puffern und das aktive Programm darin auszuführen sowie eine zugesicherte Rechenzeit, um die Verarbeitungsschritte vornehmen zu können. In einem Eingabekanal ankommende Pakete werden in der Ausführungsumgebung bearbeitet und werden dann an einen Ausgabekanal geschickt. Die dazu nötige Rechenzeit und der verwendete Speicher wird der Domäne zugeordnet. Dabei spielt es keine Rolle, ob das NodeOS, die Ausführungsumgebung oder das aktive Programm selbst eine Ressource nutzt, um den Paketfluss zu ermöglichen. Eine Domäne kapselt also letztlich alle Ressourcen, die für einen bestimmten Paketfluss genutzt werden, unabhängig davon, auf welcher Ebene sie genutzt werden.

Um eine bessere Kontrolle über die Domänen zu haben, strukturiert man sie hierarchisch. Die Wurzel bildet dabei das Betriebssystem des Knotens selbst. Auf der nächsten Ebene darunter befinden sich die verschiedenen Ausführungsumgebungen, die jeweils wieder eine eigene Domäne darstellen. Eine Domäne ist in dieser Hierarchie für die jeweils untergeordneten direkt verantwortlich. In Janos, einem Betriebssystem für aktive Knoten, wird diese Hierarchie sogar noch auf einzelne Ein- und Ausgabekanäle ausgedehnt, um eine feingranularere Kontrolle ausüben zu können [Patr01].

Das lässt sich insbesondere dann ausnutzen, wenn Domänen beendet werden sollen. Eine Domäne darf sich selbst beenden oder beispielsweise von Elternknoten wie der Ausführungsumgebung oder dem Betriebssystem beendet werden. Das kann nötig werden, wenn eine Domäne mehr Ressourcen verbraucht, als ihr zugebilligt wurde. Beim Beenden einer Domäne erhält das NodeOS alle zuvor von ihr gehaltenen Ressourcen zurück. Aus diesem Grunde haben sie eine spezielle Funktion, welche dazu beim Beenden einer untergeordneten Domäne aufgerufen wird. Sie haben hierin die Möglichkeit innerhalb einer vorgegebenen Zeit, Ressourcen freizugeben, die sie speziell für ihre untergeordneten Domänen angefordert haben. Sollte das zu

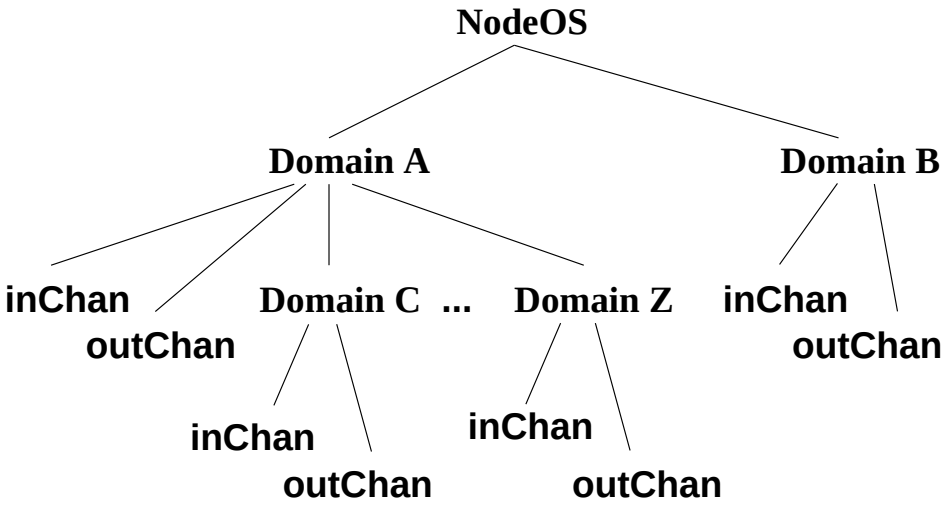

Abbildung 2: Hierarchie der Domänen in Janos.

lange dauern, wird diese Domäne einschließlich aller untergeordneter Domänen beendet. Vor allem für Ausführungsumgebungen erweist sich dieser Ansatz als sinnvoll.

Trotz dieser Hierarchie werden die Ressourcen selbst vom NodeOS und nicht von den übergeordneten Domänen vergeben. Mit diesem Ansatz wird sichergestellt, dass Domänen so viele Ressourcen allokieren können, wie sie selbst benötigen. Das soll insbesondere davon unabhängig sein, welche Ressourcen eine übergeordnete Domäne wie eine Ausführungsumgebung angefordert hat.

4.2.2 Rechenleistung

Die Rechenleistung wird durch sogenannte „thread pools“abstrahiert. Darin wird die Rechenzeit zusammengefasst, die eine Domäne in Anspruch nimmt. Beim Erzeugen einer Domäne wird eine Reihe von Eigenschaften im voraus festgelegt, durch welche sich die Nutzung der verfügbaren Rechenzeit speziell auf die Bedürfnisse der Domäne anpassen lässt. Diese Initialisierung schließt unter anderem den zu verwendenden Scheduling-Algorithmus, die maximale Ausführungszeit, die einem Thread ohne Unterbrechung gewährt werden darf, und die maximale Anzahl von Threads pro Domäne ein.

Weil eine Domäne nicht auf das aktive Programm selbst beschränkt ist, sondern logisch einen speziellen Paketfluss kapseln soll, wird die dafür gebrauchte Rechenzeit und Threads dem thread pool der Domäne angerechnet. Das bedeutet, dass auch Threads erfasst werden, die im NodeOS oder der Ausführungsumgebung ausgeführt werden, um Pakete entgegenzunehmen und an das aktive Programm weiterzuleiten. Um inkonsistente Zustände beim Paketfluss zu vermeiden, wird bei Überschreitung der Laufzeitgrenzen für einen Thread die gesamte Domäne dieses Paketflusses beendet.

Eine Sonderrolle nehmen langlebigere Threads ein, wie sie beispielsweise von einigen Ausführungsumgebungen benötigt werden. Oft sind sie keinem speziellen Paketfluss zugeordnet, und übernehmen spezielle, immer wiederkehrende Aufgaben. Eine Java Laufzeitumgebung könnte beispielsweise einen globalen Thread zur Speicherbereinigung (garbage collection) besitzen, der explizit alle anderen Threads der Laufzeitumgebung stoppen muss, um seine Funktion erfüllen zu können. Deswegen gibt es in NodeOS auch noch einige besondere Funktionen, die

es solchen Threads erlauben direkt auf das Scheduling einzuwirken und Threads zu blockieren oder die Priorität zu ändern.

4.2.3 Datenkanäle

Im Referenzmodell steht die Übertragung von Paketen genau wie bei passiven Netzen im Mittelpunkt. Domänen erzeugen unterschiedliche Datenkanäle (engl. channels), um Pakete im Netzwerk zu empfangen, versenden und weiterzuleiten.

Für jeden Kanal zum Empfangen von Daten (input channel, Abk.: inChan) muss klar festgelegt sein, welche Pakete in seinen Zuständigkeitsbereich fallen. Dazu wird beim Erzeugen angegeben, welche Protokolle und Adressbereiche inbegriffen sind. Außerdem kann eine Zeichenkette als Schlüssel angegeben werden, die in einem Paket enthalten sein muss. Der gesamte empfangene Datenstrom wird vom Betriebssystem anhand der Kriterien gefiltert, und die Pakete an die zuständigen Kanäle zur weiteren Verarbeitung geschickt. Jede Domäne stellt entsprechend Datenpuffer aus dem eigenen Speicher bereit, die mit den Paketen gefüllt werden, und besitzt eine Funktion, die ankommende Pakete behandelt.

Nach erfolgter Bearbeitung wird ein Paket in der Regel an einen Datenkanal geschickt, um es an andere Netzknoten zu versenden. Deshalb muss eine Domäne für diese Kanäle (output channel, Abk.: outChan) bestimmen, wer der Empfänger der Pakete ist und welche maximale bzw. garantierte Bandbreite gewünscht wird. Bei den meisten Netzzugangstechnologien, die heute verwendet werden, können diese Anforderungen an die Übertragungsbandbreite allerdings nicht garantiert werden.

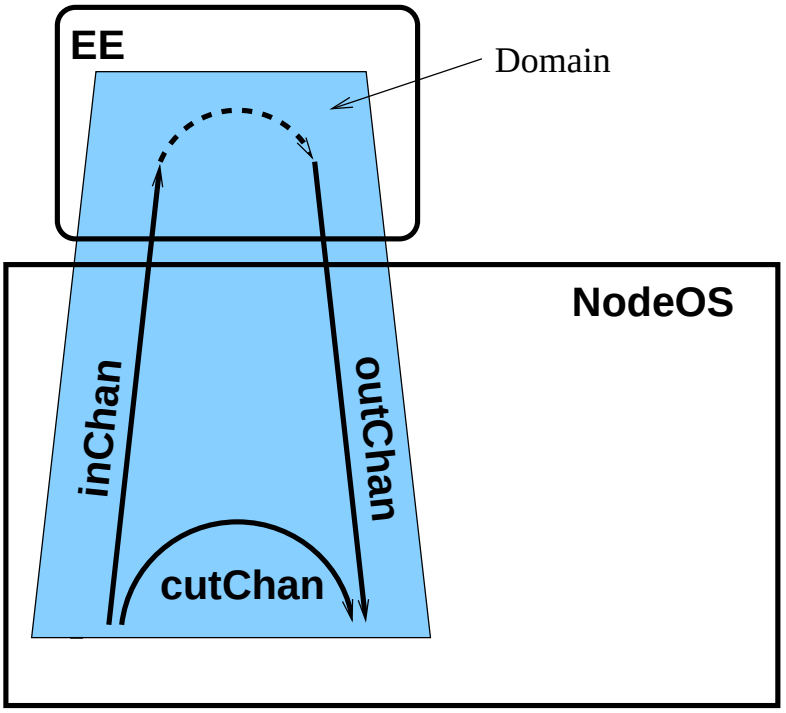

Abbildung 3: Unterstützung eines Paketfluss durch direkte Weiterleitung der Pakete.

Eine besondere Rolle nehmen die Kanäle zur direkten Weiterleitung von Paketen ein. Mit ihnen wird eine direkte Verbindung zwischen Empfangs- und Sende-Kanälen geschaffen (cut-through channel, Abk.: cutChan). So kann die Leistungsfähigkeit eines aktiven Knotens enorm

gesteigert werden und bisherige passive Netze leicht integriert werden. Optimierte Funktionen im Betriebssystem oder auf Hardwareebene sind so effizient nutzbar. Eine Anwendungsfall dafür wäre zum Beispiel ein aktives Programm, das Daten-Pakete schnell durch das NodeOS weiterleitet. Kontrollpakete werden dahingegen weiterhin an die aktive Applikation zur Verarbeitung und Steuerung des Datenflusses weitergereicht. Ressourcen, die für eine Weiterleitung gebraucht werden, rechnet man aber trotzdem der Domäne dieses Paketflusses zu.

4.2.4 Flüchtiger Speicher

Analog zu der Rechenzeit wird auch der Speicher hierarchisch in sogenannten „memory pools“ zusammengefasst. Speicher wird benötigt, um aktive Programme auszuführen, Paketpuffer zur Verfügung zu stellen, aber auch Statusinformationen für einen Datenfluss oder eine Ausführungsumgebung zu halten. Beim Speicher wird allerdings im Gegensatz zu thread pools der Speicher einer oder mehrer Domänen in einem memory pool zusammengefasst. Der Speicher wird aber trotz dieser Verwaltungsstruktur vom Betriebssystem an eine einzelne Domäne vergeben. Dadurch erhält jede Domäne genau den Speicher, den sie braucht, unabhängig davon in welcher Ausführungsumgebung sie abläuft und wie viel Speicher diese in einem thread pool bereits angesammelt hat.

Dieser Ansatz wurde gewählt, um Ausführungsumgebungen mehr Freiheit bei der Verteilung des Speichers zu geben. Gleichzeitig kann damit Speicherschutz durch mehrere unterschiedliche Adressräume realisiert werden, aber auch Speicher gemeinsam genutzt werden. Der Speicher in einem memory pool steht grundsätzlich allen beteiligten Domänen gemeinsam zur Verfügung. Dadurch kann eine Ausführungsumgebung den Speicher flexibel verwalten. Bei Domänen, die sich Speicher teilen oder auf Speicher einer anderen Domäne zugreifen wollen, können so Probleme vermieden werden, wenn eine der beteiligten Domänen beendet wird. Die Ausführungsumgebung kann Speicher einer anderen Domäne überlassen, damit diese zuvor gemeinsam genutzten Ressourcen weiterverwendet. Gleichzeitig muss sie aber dann andere ungenutzte Speicherbereiche aus dem memory pool zurückgeben, damit das Betriebssystem wieder soviel Speicher zurückerhält, wie es der beendeten Domäne insgesamt gewährt hat.

4.2.5 Persistente Speicher

Neben den vier bisher genannten Ressourcenarten wird in NodeOS auch der Zugriff auf persistente Speicher spezifiziert. Mit Dateien können aktive Applikationen Statusinformationen längerfristig speichern oder untereinander austauschen. Das Dateisystem orientiert sich am POSIX 1003.1 Standard. Jede Ausführungsumgebung erhält darin ein eigenes Verzeichnis, auf das nur sie Zugriff hat und in dem sie geschützt Dateien speichern kann. So werden die einzelnen Ausführungsumgebungen voneinander isoliert. Um trotzdem das Austauschen von Daten zu ermöglichen, sind zusätzliche Funktionen vorgesehen. Da persistenter Speicher eine Ressource ist, welche in der Praxis nicht auf allen Netzknoten zur Verfügung steht, empfiehlt die NodeOS Spezifikation als Alternative persistenten durch flüchtigen Speicher zu emulieren. Aktive Applikationen greifen so in allen Fällen über dieselbe Schnittstelle auf die Ressource zu, und es ist keine Sonderbehandlung nötig.

5 Beispiel: Ressourcenverwaltung in Janos

Janos ist ein Paket für aktive Knoten, das nach dem DARPA Referenzmodell an der Universität von Utah entwickelt wurde [Patr01]. Das Hauptziel von Janos liegt in der Ressourcenverwaltung und Kontrolle von in Java geschriebenen aktiven Programmen. Im wesentlichen

aus drei Hauptkomponenten aufgebaut besteht Janos aus einem Betriebssystem, das die NodeOS Schnittstelle implementiert, einer Java Virtual Machine und einer darauf aufsetzenden Ausführungsumgebung.

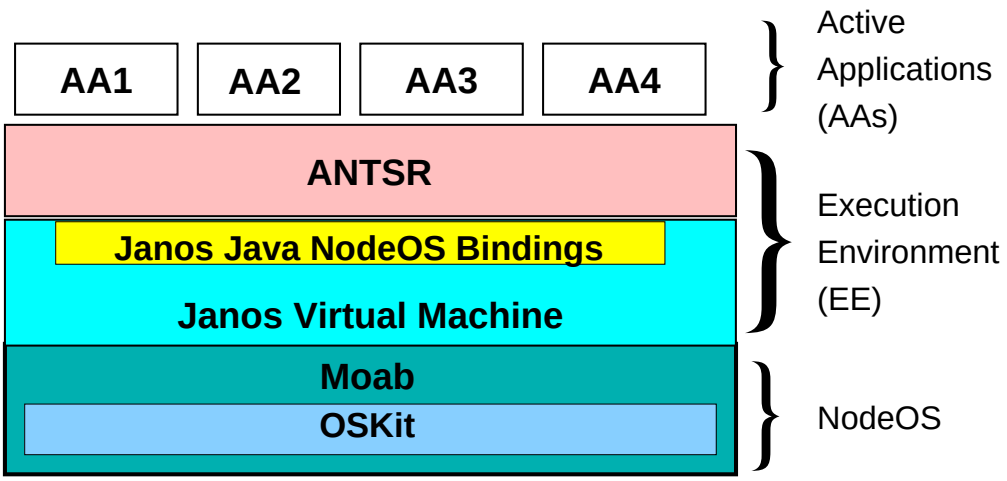

Abbildung 4: Architektur von Janos im Vergleich zum Referenzmodell.

5.1 Betriebssystem Moab

Auf der untersten Ebene befindet sich Moab, die Implementierung der NodeOS Schnittstelle. Gemeinsam mit dem OSKit, welches Dateisystem und Treiber bereitstellt, bildet Moab das Betriebssystem von Janos. Die einzelnen Komponenten von Janos sind als separat nutzbare Komponenten ausgelegt. Gegenwärtig ist Moab aber auf eine einzelne Ausführungsumgebung in Janos eingeschränkt. Neben der dadurch einfacheren Implementierung wurde dieser Ansatz gewählt, weil das Referenzmodell momentan die Frage der Sicherheit und Vertrauenswürdigkeit von Ausführungsumgebung selbst offen lässt.

Moab optimiert die Geschwindigkeit des Zugriffs auf Speicher, indem alle Puffer eine einheitliche Größe haben und in einer Speicherliste organisiert sind. Dadurch sind die Puffer untereinander leicht austauschbar, und häufiges Kopieren der Puffer wird vermieden. Das steigert die Performance des Betriebssystem, erhöht aber den gesamten Speicherbedarf vor allem wenn viele kleine Pakete übertragen werden.

Janos erweitert das Konzept der thread pools und dehnt es auf einzelne Datenkanäle aus. So können einzelne Kanäle innerhalb einer Domäne eine höhere Priorität erhalten.

5.2 JanosVM

Die JanosVM ist eine virtuelle Maschine, die Java Bytecode akzeptiert und auf Moab ausführt. Aktive Programm laufen darin unabhängig voneinander ab und haben ihren eigenen Adressraum. Der Austausch von Speicher ist nur eingeschränkt über Paketpuffer möglich. Janos gestattet aktiven Programmen den Zugriff auf die Schnittstelle des NodeOS durch eine minimale Kapselung der von Moab zur Verfügung gestellten Funktionen.

5.3 ANTSR

Die virtuelle Maschine alleine ist aber noch keine komplette Ausführungsumgebung. Um die Funktionen, die ein aktives Programm nutzen kann, geeignet einzuschränken und weitere

Sicherheitsmaßnahmen durchzusetzen, wird ANTSR genutzt. ANTSR verbirgt die kritischen Funktionen der JanosVM und grenzt die maximale Nutzung von Ressourcen je Domäne ein. Außerdem unterstützt es den integrierten Ansatz für aktive Netze, bei dem Programme und Daten in einem Paket geschickt werden. Die entsprechenden Programme werden dynamisch geladen, wenn ein neues Paket ankommt. Das sorgt gleichzeitig für Sicherheit, da so jedes Programm nur für die eigenen Pakete zuständig ist. Empfängt ANTSR ein neues Paket ohne zuständiges aktives Programm, kooperiert es mit dem Knoten, von dem das Paket geschickt wurde, und lädt das entsprechende aktive Programm, erzeugt eine neue Domäne und führt das aktive Programm aus.

6 Zusammenfassung und Ausblick

Das Beispiel Janos zeigt, das aktive Netze auch heute schon möglich sind. Gleichzeitig verdeutlicht es aber auch, dass noch erhebliche Forschungsarbeit in die Durchsetzung von Sicherheitsaspekten und wirksame Ressourcenverwaltung investiert werden muss. Neben Janos und dem NodeOS Referenzmodell gibt es allerdings bislang sehr wenige weitere Arbeiten, die sich der Ressourcenverwaltung widmen. Im Vordergrund steht an vielen Stellen momentan noch die reine Machbarkeit eines aktiven Netzes, in dem Router dieselbe Leistungsfähigkeit wie in passiven Netzen erreichen sollen. Ein großer Vorteil aktiver Netze ist aber ihre große Flexibilität. So kann nach Ausräumung der oben genannten Hauptprobleme ein langsamer Übergang von passiven zu aktiven Netzen erfolgen, bei der zeitweise beide Ansätze nebeneinander koexistieren. Ob und in welcher Form sich aktive Netze in der Praxis durchsetzen werden, ist noch offen, aber programmierbare Knoten sind heute schon vermehrt anzutreffen. Diese Entwicklung gibt Hoffnung dazu, dass sich aktive Netzwerke in Zukunft etablieren werden und eine schnellere Entwicklung der Netztechnologien erlauben werden.

Literatur

[Grou01] AN Node OS Working Group. NodeOs Interface Specification. 1 2001.

[Patr01] Jay Lepreau Patrick Tullmann, Mike Hibler. Janos: A Java-oriented OS for Active Network Nodes. 3 2001.

Abbildungsverzeichnis

Anwendungen in aktiven Netzen

Berno Kofler

Kurzfassung

Erweitert man alle, oder zumindest manche Netzwerkknoten eines Netzwerkes um zusätzliche Funktionalität, dass diese Aufgaben in allen Schichten des OSI-Modells verrichten können, ziehen netzwerkbezogene Anwendungen großen Nutzen daraus. Die Effizienz und Leistungsfähigkeit von Netzwerkmanagement, Staukontrolle, Multicasting und Caching von Daten kann bei gleichzeitiger Verringerung der Netzlast verbessert werden. Darüber hinaus erschließen sich durch den Einsatz von aktiver Netzwerktechnologie neue Anwendungsbereiche, die auf komplexe Dienste angewiesen sind, welche im Netzinneren zur Verfügung gestellt werden. Darunter fallen mobile Software-Agenten, die die Bearbeitung auf mehreren getrennten Datenbeständen in gegenseitigem Abgleich durchführen, und die so gewonnenen Informationen in aktiven Netzwerkknoten zusammenführen.

1 Einleitung

In traditionellen Netzwerken verrichten Endgeräte komplexe Aufgaben in allen OSI-Schichten, während die Vermittlungsknoten innerhalb des Netzwerkes vergleichsweise einfache Aufgaben in den Schichten 1-3 leisten. Die übermittelten Pakete enthalten, aus der Sicht des Netzwerkes, nur Daten, die in den Endsystemen interpretiert werden. Geht man nun dazu über, diese ”passiven” Netzwerke zu ”aktiven” zu erweitern, verrichten auch die Vermittlungsknoten Aufgaben in allen Schichten. In den aktiven Knoten kann nun Programmcode ausgeführt werden, der den Zustand und das Verhalten des Vermittlungsknoten ändern, oder übertragene Benutzerdaten manipulieren kann.

Die Codeübertragung zu den Netzwerkknoten kann über zwei verschiedene Ansätze erfolgen. Beim diskreten Ansatz ist die Codeübertragung unabhängig von der Datenübertragung, d.h. der Code muss vor der Anwendung auf Datenpakete auf die Zielknoten übertragen und dort gespeichert werden. Eine Kennung der Datenpakete ermöglicht die Zuordnung von Code zu Daten. Ein anderer Ansatz besteht darin, den Code in Pakete des Benutzerdatenstroms zu integrieren. Dabei wird der Code mit dem gleichen Paketstrom wie die Daten übertragen, wobei jedes Paket ein optionales Feld mit auszuführendem Code, dem sogenannten *Capsule* enthält.

Aus dem sich ergebenden Szenario wird also die Funktionsweise der Netzwerkknoten, bisher meist Router, nicht mehr durch die Firmware des Herstellers bestimmt sein. Vielmehr werden in den Knoten Laufzeitumgebungen zur Ausführung von Programmen angeboten, so dass die Netzwerkinfrastruktur dynamisch durch Programme innerhalb der Datenpakete neu konfiguriert wird. Das Verhalten der Netzwerkknoten kann pro Anwendung oder sogar pro Paket entsprechend angepasst werden.

Im folgenden Abschnitt wird gezeigt, wie aktive Netzwerke dazu benutzt werden können, die Leistung und Effizienz bestimmter Anwendungen zu verbessern. Darunter fällt das Netzwerkmanagement, insbesondere die Staukontrolle. Also Anwendungen, die Probleme im Netzinneren beheben sollen. Daher ist es aus vielerlei Gesichtspunkten sinnvoll, diese Anwendungen

direkt dort wirken zu lassen, wo ein Gerätedefekt bzw. eine Stausituation auftritt. Weiterhin ermöglicht eine aktive Netzwerktechnologie, Multicasting zuverlässig zu machen und effizienter zu implementieren. Schließlich kann das Cachen von Websites effizienter gestaltet werden und, in Verbindung mit Multicasting, seine volle Leistungsfähigkeit entfalten.

Im dritten Abschnitt sollen noch zukünftige Anwendungen vorgestellt werden, die mit aktiver Netzwerktechnologie erst möglich werden. Darunter fallen beispielsweise mobile Softwareagenten, Programme, die im Auftrag eines Benutzers oder eines anderen Programms Informationen sammeln und sich dabei selbständig durch das Netzwerk bewegen. Dabei entscheidet der Agent selbst, zu welchen Netzwerkknoten er wandert und welche Datenbanken er zur Informationsrecherche heranzieht.

2 Anwendungen

2.1 Netzwerkmanagement

Unter Netzwerkmanagement versteht man das Überwachen und Betreuen aller Netzwerkkomponenten und deren Zusammenspiel in einem Netzwerk. Darunter fallen das Fehlermanagement, das Ändern von Systemdateien auf Remotesystemen (Konfigurationsmanagement), das Abrechnungs-, Leistungs- und Sicherheitsmanagement.

Bisher existieren in der Regel in einem Netzwerk eine *Management Station* und mehrere *Devices*, wie beispielsweise Vermittlungsknoten. Die *Devices* senden ggf. Ereignisse (Traps) zur *Management Station* und umgekehrt sendet die *Management Station* Befehle und Abfragen an die *Devices* (Polling). Dabei bildet das *Simple Network Management Protokoll* (SNMP) die Grundlage dieser Kommunikation (Abb. 1).

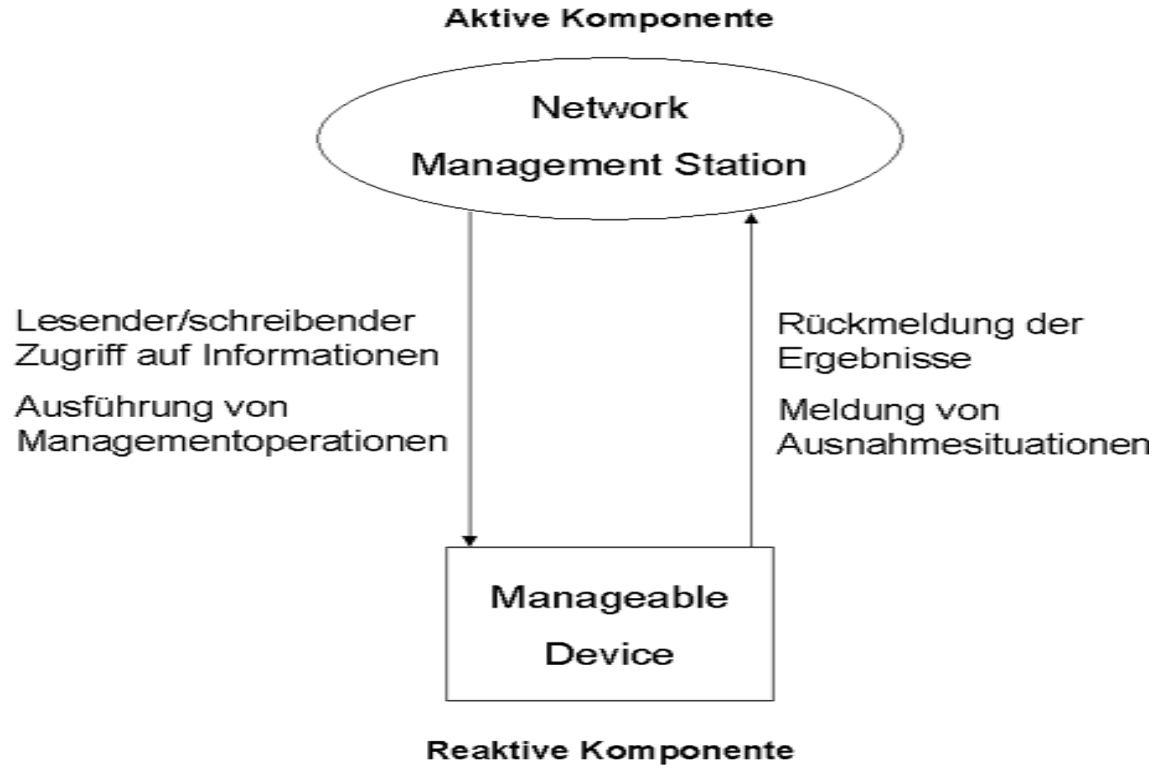

Abbildung 1: Netzwerkmanagement

In der Vergangenheit hat diese Technik gute Dienste geleistet. Mit jedoch ständig wachsenden Netzwerken, wie beispielsweise dem Internet, das exponentiellem Wachstum ausgesetzt ist,

muss das Netzwerkmanagement immer größere Mengen von Informationen und Daten bewältigen. Managementzentralen werden in zunehmenden Maße mit Meldungen von überwachten Devices überschwemmt. Dabei sind ein Großteil der übermittelten Informationen redundant. Beispielsweise wird der Managementzentrale nur mitgeteilt, dass keine Fehlersituationen im kontrollierten Teil des Netzwerkes aufgetreten sind.

Tritt nun ein Problem in einem überwachten Device auf, dann ist die Zeit, bis die Antwort der Managementzentrale im betroffenen Netzwerkknoten eintrifft, die sogenannte *round-trip delay*, signifikant groß. Die Reaktion auf die Fehlersituation kann dann schon nicht mehr zeitgemäß sein. Es ist daher offensichtlich, das Techniken des Netzwerkmanagements entwickelt werden müssen, welche zeitnäher auf Ereignisse reagieren, und gleichzeitig die durch redundante Informationen verursachte Netzlast verringern können. Es bietet sich folglich an, zu aktiven Netzwerkknoten überzugehen, um das Management direkt in das Herz des Netzwerkes, den Vermittlungsknoten, zu verlagern. Die überwachten Geräte sind also keine reaktiven Komponenten mehr, so dass Probleme schneller entdeckt, und automatisch der Managementzentrale übermittelt werden können.

Im folgenden werden mögliche Umsetzungen des Netzwerkmanagements in aktiven Netzwerken vorgestellt.

2.1.1 Smart Packets

Eine Implementierung, die aus einem Projekt bei BBN Technologies entstand, sind sogenannte *Smart Packets* [Schw]. Hierbei können Managementzentralen Programme zu den kontrollierten Netzwerkknoten senden. Dieser Ansatz hat drei Vorteile:

- Nur die zurückgelieferten Informationen, die in der Managementzentrale momentan von Interesse sind, können herausgefiltert werden, so dass einerseits die Netzlast, als auch die Menge an auszuwertender Information verringert wird.
- Viele der Managementfunktionen können in Programme transformiert werden, die dann, zu dem betroffenen Knoten geschickt, automatisch Probleme erkennen und beheben ohne, dass die Managementzentrale nochmals eingreifen muss.
- Überwachungs- und Steuerungsvorgänge werden kürzer, weil sie mittels der Übertragung eines Paketes durchgeführt werden, anstatt durch Folgen von *Set*- und *Get*-Befehlen.

Netzwerkmanagement- und Überwachungsprogramme erstellen *Smart Packets*, die in *Active-Network-Encapsulation-Protocol*-Rahmen (ANEP-Frames) gebettet sind und senden diese an den ANEP-Dämon (Abb. 2). Dieser hat zum einen die Aufgabe, *Smart Packets* zu senden und zu empfangen, und andererseits die virtuelle Maschine zur Ausführung des empfangenen Programmcodes zur Verfügung zu stellen. Das in *Smart Packets* eingebettete Programm kann entweder nur im Zielknoten (*end-to-end-mode*) oder auch in allen dazwischenliegenden Knoten (*hop-by-hop-mode*) ausgeführt werden. Dieses Programm kann beispielsweise Direktiven enthalten, dem Netzwerkmanagement-Programm von jedem Knoten Resultate zurückzusenden oder dazwischenliegende Knoten von der Ausführung auszuschließen.

Ein *Smart Packet* besteht aus einem Header gefolgt von dem ausführbaren Code. Das *Smart Packet* ist wiederum in ein ANEP-Paket eingebettet, welches wiederum in einem IP-Paket gekapselt ist. Zudem wird ein Tag im Optionsfeld des IP-Headers zur Kennzeichnung dieser IP-Pakete gesetzt, so dass ein Router in einem aktiven Netzwerk erkennen kann, ob das Paket ausführbaren Code enthält. Unterstützt der Router keine aktiven Netzwerke, ignoriert

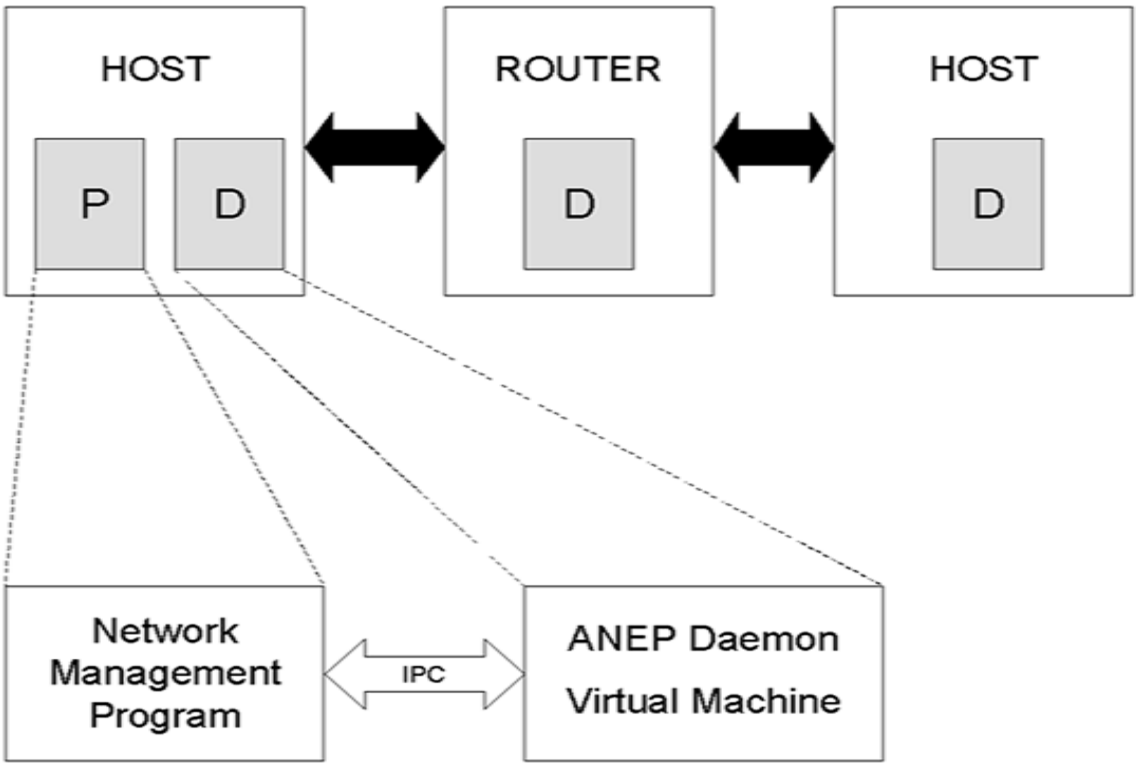

Abbildung 2: ANEP Dämon

er den Tag im Optionsfeld und leitet das Paket nur weiter. Ansonsten wertet der Router den ANEP-Rahmen aus und, falls er *Smart Packets* unterstützt, führt er den Programmcode aus.

Es gibt vier Typen von *Smart Packets*:

- Programmpakete: Sie enthalten den Code, der in entsprechenden Netzwerkknoten ausgeführt werden kann.
- Datenpakete: Das Resultat der Programmausführung ist in ihnen enthalten, um es dem Managementprogramm zurückzuliefern.
- Nachrichtenpakete: Sie beinhalten Nachrichten und Zustandsinformationen.
- Fehlerpakete: Sie liefern Fehlerzustände, wenn der Transport eines Programmpaketes oder die Ausführung desselben fehlgeschlagen ist.

Zur Programmierung der *Smart Packets* wurden die zwei Programmiersprachen *Sprocket* und *Spanner* entwickelt. *Sprocket* ist eine Hochsprache mit eingebauter Unterstützung für das Netzwerkmanagement, und *Spanner* ist eine CISC-Assemblersprache, in die *Sprocket* übersetzt wird. Ein Entwurfsziel war es, Netzwerkmanagementprogramme sehr kompakt, in weniger als einem Kilobyte codieren zu können, da ein vollständiges Managementprogramm nur aus einem Paket bestehen soll.

2.1.2 Management durch Delegation

Im *Network Management by Delegation*-Ansatz der Columbia University können sogenannte *management processing functions* dynamisch zu Netzwerkkomponenten delegiert und dort lokal ausgeführt werden [Psou]. Dabei kommen folgende Komponenten zum Einsatz:

- *Elasitc Server*: Ein multithreaded Prozeß, der die Ausführung von *delegated agents* ermöglicht. Die Laufzeitumgebung unterstüzt das Übersetzen und Linken des Programmcodes, eine ferngesteuerte Kontrolle der Agenten sowie *inter-agent-* und *agent-to-process*-Kommunikation. Als Programmiersprachen unterstützt der *Elastic Server* C, C++ und TCL.

- *Delegated Agents*: Sie werden dynamisch zu einem entfernten *Elastic Server* gesendet. Sie stellen eine Ausprägung von mobilen Softwareagenten dar, die im Abschnitt 3 näher betrachtet wird.

- Delegations-Protokoll: Dies koordiniert das Versenden der *Delegated Agents* zu den *Elastic Servern* und überwacht deren ordnungsgemäße Ausführung.

Netzerkmangement durch Delegation verringert einerseits die verwendete Bandbreite, die für Managementzwecke benötigt wird, andererseits können Entwickler zeitnäher Managementregeln ändern. Dieser Ansatz verbessert die räumliche und zeitliche Verteilung der Managementfunktionen.

2.1.3 Darwin-Projekt

Beim Darwin-Projekt der Carnegie Mellon University werden sogenannte *Delegates* zu Routern geschickt und können über eine API das Ressourcenmanagement des jeweiligen Routers übernehmen [Psou]. Die API bietet u.a. folgende Dienste an:

- Datenströme können aufgesplittet oder zusammengefasst werden.

- Routingregeln können verändert werden.

- Delegates können Nachrichten senden und empfangen.

2.2 Staukontrolle

Eine andere Problematik, die auch in Zukunft aufgrund des steigenden Netzwerkverkehrsaufkommens nicht zu vernachlässigen sein wird, sind Netzwerküberlastungen. Daher ist es notwendig, effiziente Algorithmen zu finden, die eine rasche Erkennung und rechtzeitige Beseitigung der Stausituation zulassen, um Paketverluste zu vermeiden.

Die Netzüberlastungen treten im Netzinneren, weit entfernt von Endsystemen auf. Somit kann relativ viel Zeit vergehen, bis die Stausituation vom Empfänger entdeckt wird und das Protokoll einen selbstregulierenden Mechanismus (beispielsweise *slow start/congestion control* bei TCP) einsetzen kann, um die Stausituation zu mildern. Einerseits verstärkt sich in dieser Reaktionszeit die Überlastung des Netzwerkknotens, andererseits kann nach dieser Zeit die Überlastung bereits aufgelöst sein, so dass ein regulierender Eingriff nicht mehr nötig ist.

Die Staukontrolle fällt unter den Bereich des Netzwerkmanagements. Dadurch treffen alle im vorangegangenen Abschnitt genannten Verbesserungen durch Einsatz aktiver Netzwerktechnologie auch hier zu. Einige Beispiele sollen das verdeutlichen:

- Ein aktiver Netzwerkknoten kann die zur Verfügung stehende Bandbreite kontrollieren und dementsprechend die Datenflussrate überwachen.

- Im Fall von mehreren Datenflüssen mit verschiedenen Anforderungen an die Staukontrolle, kann ein aktiver Knoten die jeweilige Rate eines jeden Datenflusses unter Beachtung der Gesamtbandbreite regulieren. Zudem ist es möglich, dynamisch auf Bandbreitenanforderungen von Datenflüssen zu reagieren.

- Steht ausreichende Rechenkapazität im aktiven Netzwerkknoten zur Verfügung, so kann im Falle einer Stausituation, ein Medienstrom in einen Strom schlechterer Qualität in Echtzeit transformiert werden, um weniger Bandbreite zu beanspruchen.

- Das selektive Verwerfen von Paketen läßt sich mit einem MPEG-Videostrom als Beispiel verdeutlichen. Geht innerhalb des Stromes ein I-Frame (Intraframe-codiertes Bild) verloren, besteht keinerlei Notwendigkeit mehr, die P- (unidirektional präzidiertes Bild) und B-Frames (bidirektional präzidiertes Bild), die ja von dem verlorengegangen I-Frame abhängen, weiterzuleiten [Zota01].

Wie sieht nun eine konkrete Implementierung der Staukontrolle in aktiven Netzwerken aus? Am Georgia Institute of Technology geht man jenem Ansatz nach, eine endliche Menge von Funktionen zu definieren, die in einem aktiven Netzwerkknoten durch einen sogenannten *active processor* ausgeführt werden können [Psou]. Die Durchführung von Berechnungen mit Hilfe dieser Funktionen schließen die Zustandsinformationen, die im jeweiligen Knoten persistent abgelegt sind, mit ein. Die Zustandsinformationen wandern nur zwischen Knoten, wenn diese Informationen in Pakete eingebettet wurden. Jedes Paket beinhaltet in seinem Header eine Nummer, welche die Funktion identifiziert, die ausgeführt wird und eine Menge von Identifizierungsmerkmalen, die jene zur Berechnung der Funktion benötigien Zustandsinformationen bestimmen. Diese Einheit wird *active processing control information* (APCI) genannt.

Trifft nun ein Paket in einem Knoten ein, wird zunächst der Zielport bestimmt und, falls eine ACPI vorhanden ist, diese dem *active processor* übergeben. Anderenfalls wird das Paket weitergeleitet. Ist nun eine APCI vorhanden, wird die darin spezifizierte Funktion ausgeführt. Je nach ausgeführter Funktion wird der Zustand des Knotens aktualisiert oder das modifizierte Paket weitergeleitet.

Setzt man diesen Ansatz in Bezug zu den oben genannten Beispielen, ergeben sich folgende Hauptfunktionen:

- *Buffering and Rate Control*: Mit Hilfe dieser Funktionen überwacht der *active processor* die vorhandene Bandbreite und die Datenraten und kann so Auswertungen auf Basis der gesammelten Daten machen.

- *Unit-Level Dropping*: Einheiten, die aus mehreren Paketen bestehen, werden verworfen, falls Pakete verlorengegangen sind, die in einer Abhängigkeit zueinander standen (beispielsweise in einer *Group of Pictures* (GoP) bei MPEG).

- *Media Transformation*: Diese Funktionen ermöglichen die Transformation der Daten in eine Form, die beim Auftreten eines Netzwerkstauus weniger Bandbreite benötigen.

Dieser einfache Ansatz hat, obwohl er die Möglichkeiten aktiver Netzwerke nicht vollständig ausnutzt, einige Vorteile. Es reicht aus, wenn nur einige Netzwerkknoten mit aktiver Netzwerktechnologie ausgestattet werden, herkömmliche Router leiten diese Pakete einfach weiter. Weiterhin ist die Komplexität dieses Ansatzes überschaubar, weil die Pakete einerseits keinen ausführbaren Code beinhalten und in den Knoten nur eine endliche Anzahl von Funktionen implementiert ist.

2.3 Multicasting

Viele Kommunikationsanwendungen im Internet gehen davon aus, dass ein Teilnehmer (der Sender) Daten erzeugt, die zu vielen anderen Teilnehmern (den Empfängern) übertragen werden. So gibt es viele Szenarien, bei denen Tausende von Teilnehmern dieselben Informationen benötigen. Ein Sender, der Datenpakete, die sogenannten Multicast-Datagramme, an eine Menge von Empfängern versenden will, sendet diese Daten an eine zuvor festgelegte Gruppe. Die Empfänger müssen Mitglieder dieser Gruppe sein, die durch eine Klasse D IP-Adresse identifiziert wird. Die Anzahl der Mitglieder einer Gruppe kann sich fortwährend ändern. Die Übertragung geschieht heutzutage über den Multicast Backbone, einem Overlay-Netzwerk des Internets. Die Informationen über die Teilnehmer werden nicht beim Sender, sondern im Netz gehalten, und zwar in mutlicastfähigen Routern. Über jede Netzwerk-Verbindung geht jeweils nur eine Kopie der Multicast-Pakete, lediglich bei Verzweigung der Netzstruktur werden Kopien in den Routern erzeugt, sollte die Lage der Empfänger dies erfordern.

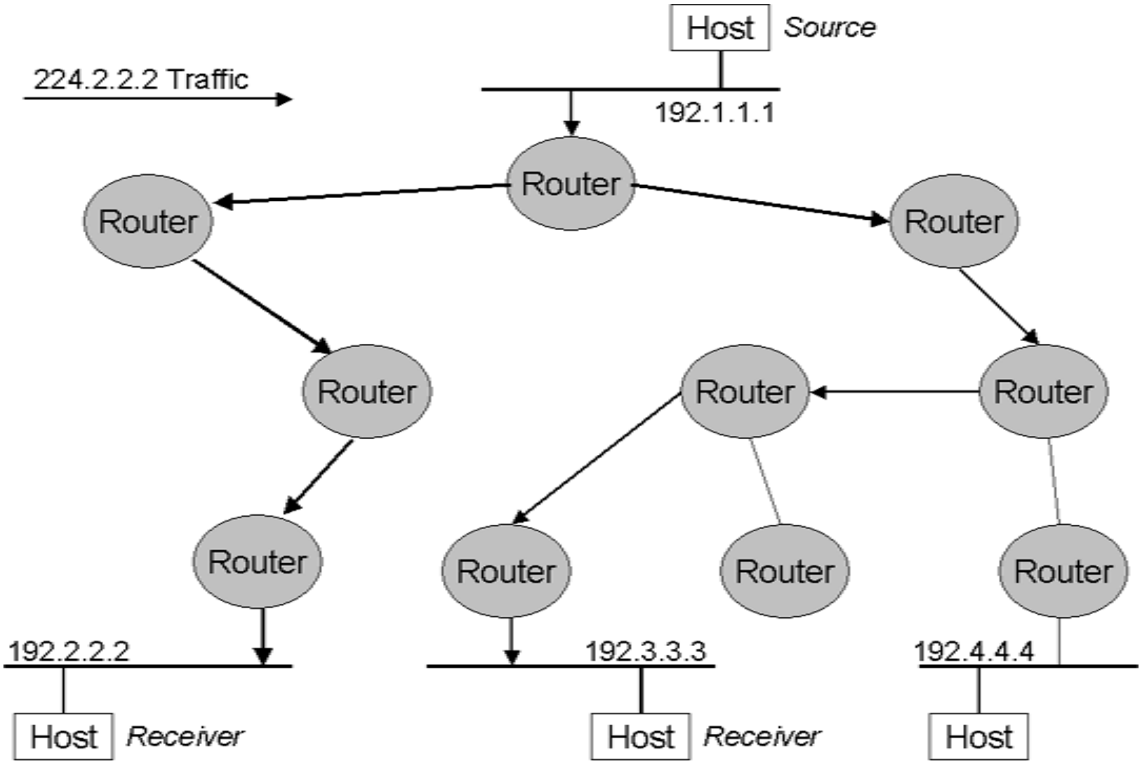

Abbildung 3: IP-Multicasting

Das Versenden von Multicast-Paketen ist, wie auch im Fall von UDP, nicht zuverlässig. Es wird also nicht kontrolliert, ob alle gesendeten Pakete fehlerfrei und in der richtigen Reihenfolge bei allen Empfängern ankommen. Nicht zuverlässige Protokolle sind jedoch in ihrer Anwendbarkeit stark eingeschränkt, da in sehr vielen Anwendungen, ein fehlerfreier Empfang der Daten vorausgesetzt wird. Im Falle des Unicasting steht hier TCP zur Verfügung, das eine Bestätigung (*Acknowledgement*, kurz:ACK) für den fehlerfreien Empfang von Daten erwartet. Wird eine solche Bestätigung nicht gegeben, werden die Daten erneut gesendet.

Dieses Vorgehen ist bei einem zuverlässigen Multicast-Protokoll nicht ohne weiteres möglich, da insbesondere in größeren Gruppen, das Versenden von vielen *Acknowledgements* an einen Rechner zu einer Überlastung des Netzwerkes führen würde, der sogenannten *acknowlegement implosion*. Eine andere Möglichkeit besteht darin, die fehlerfreie Übertragung über negative *Acknowledgements* (NACKs) zu sichern. Ein Endsystem, das über eine laufende Nummer feststellt, dass ein Paket fehlt, wird ein NACK an den Sender schicken, der die fehlenden

Daten nochmals versendet. Allerdings kann es auch diesem Fall zu einer NACK implosion in den Netzwerkknoten kommen [Psou]. Ein weiteres Problem von zuverlässigen Multicast-Protokollen ist der Paketverlust aufgrund von Überlastungen der Router. Die Paketverluste führen bei zuverlässigen Protokollen zu einem starken Anstieg neu versendeter Pakete, welche die Überlastung der betroffenen Router weiter erhöhen, ein Kreislauf, der zum Zusammenbruch von Teilen des Netzes führen kann.

Zusammengefasst müssen folgende Ziele erreicht werden:

- Verhinderung von *NACK implosions*.
- Minimierung von Übertragungswiederholungen.
- Vermeidung von unnötigen Übertragungswiederholungen.
- Robustheit gegenüber Veränderungen der Gruppengröße und -struktur.

Am MIT wurde ein Verfahren entwickelt, das diesen Anforderungen gerecht wird und dabei die Vorteile eines aktiven Netzwerkes ausnutzt [LeGT98].

2.3.1 Active Reliable Multicast (ARM)

Vorausgesetzt wird, dass die Pfade im Multicast Routingbaum den umgekehrten Unicast Routingpfaden entsprechen: ARM verlässt sich darauf, dass NACKs, die per Unicast vom Empfänger zum Sender geschickt werden, durch die gleichen Vermittlungsknoten laufen, wie die entsprechenden Multicastpakete vom Sender zum Empfänger. Im folgenden werden die Funktionen der aktiven Knoten genauer beschrieben. Das Protokoll ist so konzipiert, dass nicht alle Knoten aktiv sein müssen - im Extremfall, wenn kein aktiver Knoten vorhanden ist, wird aber weder *NACK implosion* verhindert noch gibt es eine isolierte Fehlerbehebung.

Zunächst werden in ausgezeichneten Netzwerkknoten, die an einer Multicastsitzung beteiligt sind, Multicastpakete zwischengespeichert. Die Knoten unterhalten hierfür einen Cache. Der Header der Multicastpakete wird hierfür um eine *cache time-to-live* (Cache TTL) erweitert. Diese gibt an, wie lange das Paket sinnvollerweise im Speicher des Knotens gehalten werden soll. Diese Zeit hängt davon ab, wie schnell ein aktiver Knoten mit einem NACK des am weitesten entfernten Empfängers rechnen kann, die wiederum von der Verzögerung, der sogenannten *round trip time* (RTT), zwischen Knoten und Empfänger abhängig ist. Im Cache wird dann das Paket selbst, die Cache TTL, die Gruppenadresse, die Adresse des Senders und die Sequenznummer abgelegt. Sendet ein Empfänger ein NACK, kann ein Router, der die benötigten Pakete vorhält, jenes abfangen und selbst die Übertragungswiederholung durchführen. Studien haben gezeigt, das die meisten Paketverluste an den Eckpunkten von Netzwerken auftreten. Demzufolge reicht es aus, Pakete nur in strategisch wichtigen Netzwerkknoten zwischenzuspeichern, die beispielsweise lokale Netzwerke oder eine Luftschnittstelle anbinden. Anstatt alle weitergeleiteten Pakete zwischenzuspeichern, können auch nur sogenannte Reparaturpakete in den Knoten vorgehalten werden. Das ist besonders sinnvoll, wenn nur wenig Speicherplatz im Knoten zur Verfügung steht oder häufig Übertragungswiederholungs-Anfragen aus entfernten Teilen des Netzwerkes eintreffen.

Für die Unterdrückung von NACKs und die lokale Fehlerbehebung müssen im aktiven Knoten zusätzlich zu den Caches und der normalen Multicast Routinginformation weitere Informationen gehalten werden, sogenannte NACK- und REPAIR- Records.

- Ein ARM NACK Paket enthält die Adresse des Empfängers, der das NACK erzeugt hat, die Adresse des Senders, die Gruppenadresse, die Sequenznummer des fehlenden Pakets, einen NACK-Zähler und eine Cache TTL. DER NACK-Zähler wird vom Erzeuger des NACK bei jedem weiteren gesendeten NACK für dieses Paket inkrementiert.
- Ein REPAIR-Record für ein bestimmtes Datenpaket enthält einen Vektor, der für jeden Link anzeigt, ob über diesen bereits eine Antwort auf ein NACK gesendet wurde und welchen Wert der dazugehörige NACK-Zähler hatte.
- Ein NACK-Record wird im ARM Router angelegt, wenn zu einem empfangenen NACK weder das entsprechende Datenpaket im Cache, noch ein entsprechender NACK- oder REPAIR-Record vorhanden ist. Der NACK-Record enthält den maximalen NACK-Zähler, der für dieses Paket empfangen worden ist und eine sogenannte *Subscription Bitmap*, die anzeigt, auf welchen Verbindungen NACKs für dieses Paket eingegangen sind. Diese Subscription Bitmaps bewirken, dass verlorene Pakete möglichst nur an die Empfänger der Multicastgruppe ausgeliefert werden, denen diese Pakete auch fehlen. Findet ein ARM Router zu einem empfangenen Paket einen entsprechenden NACK Record, so wird das Paket nur auf den in der *Subscription Bitmap* angegebenen Links weitergeleitet. Auf diese Weise wird sichergestellt, dass nur die Empfänger, die ein NACK verschickt haben, das Paket bekommen.

Empfängt nun ein ARM Router ein NACK auf Link l_1 können folgende Zustände unterschieden werden [HoCh99]:

- Existiert schon ein REPAIR-Paket für dieses Paket und ist der NACK-Zähler für Link l_1 größer oder gleich dem des eingetroffenen NACKs, so wird es verworfen.
- Ist kein REPAIR-Record vorhanden, aber das gewünschte Paket im Cache vorhanden, so wird das Paket auf Link l_1 gesendet und ein entsprechender REPAIR-Record angelegt.
- Ist weder ein REPAIR-Record, ein NACK-Record, noch das Datenpaket im Cache vorhanden, so wird das NACK in Richtung Sender weitergeleitet, ein entsprechender NACK-Record erstellt und das für den Link l_1 zuständige Bit in der *Subscription Bitmap* gesetzt.
- Gibt es schon einen NACK-Record für dieses Paket und ist der NACK Zähler größer oder gleich dem des eingetroffenen Pakets, so wird das für Link l_1 zuständige Bit in der *Subscription Bitmap* gesetzt und das NACK verworfen.

Simulationen haben gezeigt, dass es bereits ausreicht, weniger als 50 Prozent aller Router mit aktiver Netzwerktechnologie auszurüsten, damit ARM seine volle Funktionsfähigkeit entfalten kann [Psou].

2.4 Caching

In gleichem Maße wie die Anzahl der Internetknoten wächst, steigen auch die Datenmengen die über die Netze transportiert werden. Insbesondere die interkontinentalen Verbindungen stoßen oft an die Grenzen ihrer Belastbarkeit. Diese Entwicklung ist insbesondere mit dem Aufkommen des WWW noch beschleunigt worden. Durch wiederholtes Anklicken von Hyperlinks in HTML-Dokumenten werden beliebig oft die gleichen Daten angefordert. Durch den Einsatz von WWW-Caching lässt sich die Zahl solcher Übertragungen verringern, indem häufig angeforderte Seiten gecached werden. Das verringert in erster Linie die Netzlast und natürlich auch die Zeit bis die angeforderten Informationen beim Client verfügbar sind.

Bisher wurden WWW-Inhalte lokal im Client und an administrativen Grenzen in Proxies zwischengespeichert. Im Falle von aktiven Netzwerkknoten, können darüberhinaus auch diese zur Zwischenspeicherung von Daten benutzt werden. Die erste Frage stellt sich nach der Platzierung der Caches - üblicherweise an strategisch wichtigen Punkten:

- In Transit-Knoten (*transit-only caching*), weil ein großer Teil aller Netzwerkpfade durch diese Vermittlungsknoten laufen.
- In Knoten, die ein *Stub*-Netzwerk anbinden, weil jene passiert werden müssen, wenn ein Rechner den Rest des Netzwerkes erreichen möchte.

Schöpft man die Möglichkeiten aktiver Netzwerke weiter aus, so können nicht nur die zwischengespeicherten Informationen, sondern auch die Caches selbst im Netzwerk wandern. Bei diesem Ansatz entscheiden also die aktiven Netzwerkknoten selbst, ob weitergeleitete Daten zwischenspeichert werden. Dabei sollten die Knoten Informationen, die in Kürze von nahegelegenen Clients angefordert werden, zwischenspeichern und sich gleichzeitig mit anderen Knoten koordinieren, so dass beispielsweise in unmittelbaren Nachbarknoten nicht dieselben Informationen gecacht werden.

Weiterhin können die Programme für die Erstellung dynamischer Internetseiten in aktiven Knoten nahe den Clients gehalten und ausgeführt werden.

Diese Ideen wurden in einem Projekt am Georgia Institute of Technology innerhalb von zwei unterschiedlichen Ansätzen umgesetzt [BhCZ].

2.4.1 Modulo Caching

Zunächst wird ein sogenannter *Cache Radius* definiert, der die Entfernung zweier Caches in Hops angibt. Die Caching-Strategie benutzt den Cache Radius folgendermaßen: Fordert ein Client Informationen von einem Server an, werden die Objekte in jenen Vermittlungsknoten auf dem Pfad Server-Client gecacht, die im Abstand des *Cache Radius* lokalisiert sind. Die Objekte sind sodann in konzentrischen Ringen um den Server angeordnet, welche entsprechend der dazwischenliegenden Hops durchnummeriert werden (Abb. 2.4.1).

Der *Cache Radius* ist ein Parameter, der global, für jedes Objekt, oder in verschiedenen Netzwerkbereichen lokal gesetzt werden kann. Der Mechanismus, der zur Implementierung dieser Caching-Strategie in jedem Knoten eingesetzt wird, ist eine einfache Modulo-Funktion. Die vom Server gesendeten Objekte enthalten einen Hop-Zähler, der mit der Sequenznummer des Objektes modulo des *Cache Radius* initialisiert wird. In jedem von dem Paket durchlaufenen Knoten wird der Hop-Zähler inkrementiert. Ergibt Hop-Zähler modulo *Cache Radius* den Wert 0, wird das Objekt in diesem Knoten zwischengespeichert.

2.4.2 Caching with Lookaround

Der Größenunterschied zwischen einem Objekt und einem Verweis darauf, beispielsweise einer IP-Adresse, ist bedeutend groß. Diese Tatsache kann dazu genutzt werden, in jedem Netzwerkcache ein Teil des Speichers dazu zu verwenden, Verweise auf Objekte anzulegen, die in einer Umgebung des Knotens zwischengespeichert sind. Dazu hält jeder Knoten eine in bestimmten Intervallen aktualisierte Tabelle vor, die die zwischengespeicherten Objekte der Nachbarknoten enthält. Jeder Netzwerkcache kann in logische Zonen, sogenannte *levels* eingeteilt werden: *Level* 0 enthält alle Objekte, die im Knoten selbst lokal abgelegt sind. *Level*

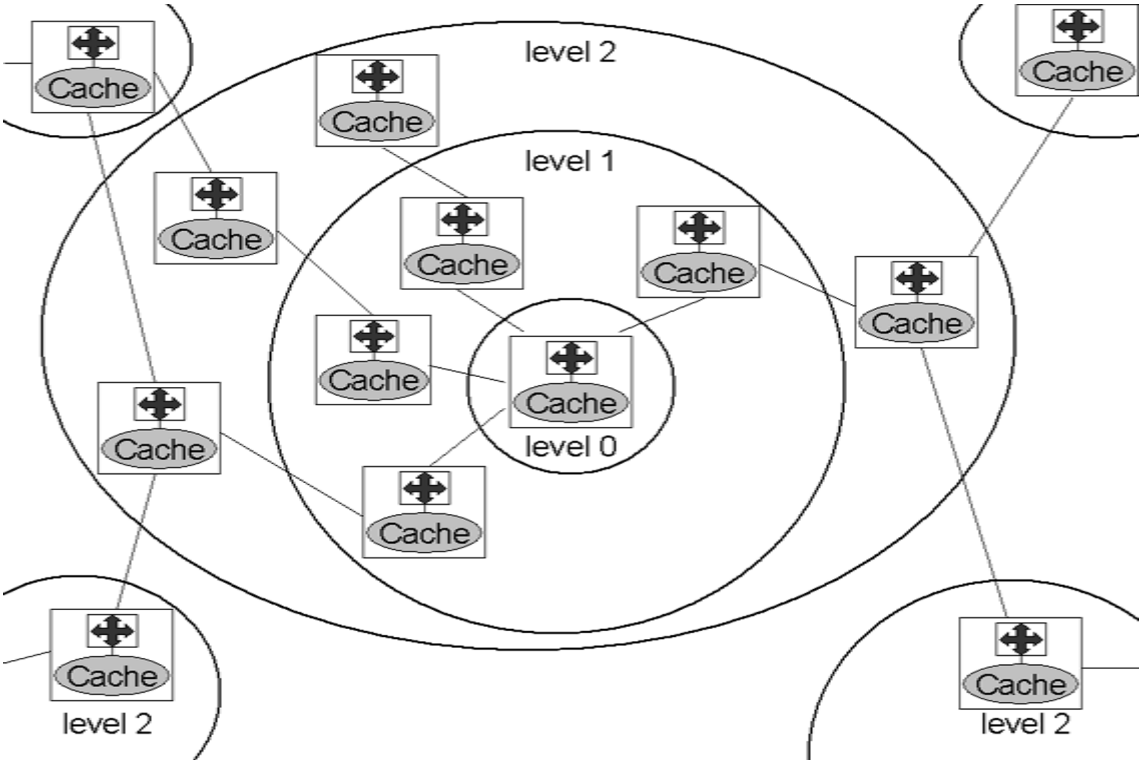

Abbildung 4: Modulo Caching

1 enthält Verweise auf Objekte, die in einem Netzwerkknoten einen Hop entfernt zwischengespeichert sind, *Level* 2 enthält Verweise auf Objekte, die in Knoten in einem Abstand von zwei Hops entfernt gespeichert sind, usw. . Die Anzahl der *levels* ist dabei geeignet zu wählen.

Trifft eine Anfrage an einem Knoten ein, werden die *levels*, beginnend mit *Level* 0 durchsucht. Stellt sich dabei heraus, dass sich das gewünschte Objekt in einem Nachbarknoten befindet, wird die Anfrage an diesen Knoten weitergeleitet, ohne die ursprüngliche Zieladresse zu ändern. Kann nämlich das gewünschte Objekt in dem referenzierten Knoten nicht mehr gefunden werden, wird die Anfrage an den Server weitergeleitet, der ursprünglich angegeben wurde. Damit ist dieses Verfahren ausfallsicher und kompatibel zu heterogenen Netzwerken, die sowohl aus passiven als auch aktiven Netzwerkknoten bestehen.

Beide hier vorgestellten Verfahren reduzieren die Latenzzeit, die Verringerung von benötigter Bandbreite steht im Hintergrund. Daher ist es wohl sinnvoll, traditionelle Verfahren und selbstorganisierende Caches zu kombinieren. Beispielsweise sorgt ein Proxy-Server einer Domäne dafür, dass weniger Bandbreite benötigt wird, während selbstorganisierende Caches die Latenzzeit in Weitverkehrsnetzen reduzieren.

3 Mobile Softwareagenten

Im Gebiet der Verteilten Systeme versteht man unter einem mobilen Agenten, eine Softwarekomponente, die fähig ist, sich von einem Host oder einem aktiven Netzwerkknoten zu einem anderen autonom zu bewegen. Diese Programme befördern sich, d.h. den Code und den aktuellen Zustand, direkt zu den Ressourcen. Das verbessert oft die Leistung, da die Agenten dorthin wandern, wo sich die Daten befinden, anstatt die Daten zu den bewegt Agenten bewegt werden [Sund].

Durch die Technologie der mobilen Agenten können zahlreiche Probleme gelöst werden. Einige Beispiele sollen hier angeführt werden:

- Bandbreitenprobleme: Eine Transaktion oder Anfrage zwischen einem Client und einem Server kann mehrere Übertragungen benötigen, welche bei großer Anzahl von Clients und Transaktionen zu Engpässen führen kann. Durch den Entwurf von Agenten, die Anfragen oder Transaktionen behandeln, kann ein Teil der Bandbreite wieder freigeben werden. Diese Agenten werden direkt vom Client zum Server geschickt. Es müssen nur noch die Agenten versendet werden, nicht aber die Zwischenergebnisse und Informationen.

- Unzuverlässige Netzwerkverbindungen: Bei zahlreichen Netzwerkanwendungen muss die Netzwerkverbindung aufrechterhalten werden, um ein korrektes Ausführen einer Transaktionen oder einer Anfrage zu ermöglichen. Wenn die Verbindung getrennt wird, muss der Client oft die Transaktion oder die Anfrage erneut starten, wenn sie überhaupt neu gestartet werden kann. Die Technik der Agenten erlaubt einem Client, wenn die Netzwerkverbindung noch besteht, einen Agenten zu senden, der die Transaktion/Anfrage im Netzwerk bearbeitet. Der Client kann dann die Verbindung trennen. Der Agent wird die Transaktion/Anfrage alleine bearbeiten und das Resultat dem Client präsentieren, wenn die Verbindung wieder besteht.

- Künstliche Intelligenz: Anwendungen für Intelligente Software-Agenten (ISAs), die autonom arbeiten, beinhalten Problembereiche, die menschliche Intelligenz benötigen. Darunter fallen

  - Logisches Folgern und Schlüsse ziehen
  - Erkennen von Mustern
  - Lernen
  - Anpassungsfähigkeit

  Autonome Operationen erlauben ISAs schon jetzt, riesige Datenmengen zu verarbeiten, die für Menschen nicht erfassbar wären. Heute werden ISAs erfolgreich in mehrfachen Anwendungen eingesetzt. Einige davon sind:

  - Sammeln von Daten
  - Filtern von Daten
  - Erkennen von Mustern
  - Präsentation von Daten
  - Planen und Organisieren

Die Zugänglichkeit von Ressourcen, die über das Internet verfügbar sind, ist rapide angestiegen. Die Informationsflut kann erfolgreich von intelligenten Software-Agenten verwaltet werden. Am wirksamsten durch das Umwandeln von Rohdaten in brauchbares Wissen und passende Aktionen.

4 Conclusion

Es wurde gezeigt, bei welchen Anwendungen sich aktive Netzwerke als vorteilhaft erweisen. Darunter fallen Netzwerkmangement, Staukontrole, zuverlässiges Multicasting sowie Caching von Webinhalten. Dazu wurden aktuelle Projekte vorgestellt: Smart Packets bei BBN Technologies, Netzwerkmangement durch Delegation an der Columbia University, das Darwin-Projekt an der Carnegie Mellon University, Active Reliable Mutlicast (ARM) am MIT sowie

Wide-Area-Network-Caches am Georgia Institute of Technology. Schließlich wurde noch eine neue Anwendung vorgestellt, die erst mit aktiver Netzwerktechnologie möglich wird, die mobilen Softwareagenten.

Ob die hier vorgestellten Projekte jedoch auch außerhalb den Testzenarien bestand haben werden, wird sich erst zeigen, wenn diese Verfahren in größeren Netzwerken getestet und eingesetzt werden. Weiterhin muss zunächst noch überprüft werden, ob die wesentlich erhöhte Komplexität der Netzwerkknoten im richtigen Verhältnis zu dem Nutzen für die hier vorgestellten Anwendungen steht.

Literatur

[BhCZ] Samrat Bhattacharjee, Kenneth L. Calvert und Ellen W. Zegura. Self-Organizing Wide-Area Network Caches. Technischer Bericht, Networking and Telecommunications Group, Georgia Institute of Technology, Atlanta, GA 30332.

[HoCh99] H.W. Holbrook und D.R. Cheriton. *IP Multicast Channels: Epress Support for Large-scale Single-source Applications.* ACM Sigcomm, Cambridge, Massachusetts. 1999.

[LeGT98] H. Lehmann, Stephan J. Garland und D. L. L. Tennenhouse (Hrsg.). *Active Reliable Multicast*, San Francisco, CA, 1998. IEEE INFOCOM.

[Psou] Konstantinos Psounis. Active Networks: Applications, Security, Safety and Architectures. Technischer Bericht, Stanford University.

[Schw] Beverly Schwartz. Smart Packets for Active Networks. Technischer Bericht, BBN Technologies, 10 Moulton St., Cambridge MA 02138.

[Sund] Todd Sundsted. An Introduction to Agents. Technischer Bericht, Internet Workshopm, Juni 1998, http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html.

[Zota01] Dr. Volker Zota. Eingedampfte Bilderströme. *ct magazin* (10), Mai 2001, S. 130,131.

Abbildungsverzeichnis

Einführung eines neuen Dienste in aktiven Netzen

Yining Zhao

Kurzfassung

Aktive Netze(*Active Netzworks*) repräsentiert einen signifikanten Schritt in der Revolution der *packet-switched* Netze, von dem traditionalen *packet-forwarding* Mechanismus zu mehr generellen Funktionalitäten zur Unterstützung dynamischer Kontrolle und Modifikation des Netzverhaltens. Dynamische Netzdienste (*network service*) machen diese Innovation durch neue aktive Technologien möglich. In diesem Artikel werden die heutige Anwendung der Dienste in aktiven Netzen vorgestellt sogar diskutiert, dadurch daß wie neue Dienste in aktiven Netzen eingeführt werden.

1 Einleitung

In allgemeinen waren die unveränderten passiven Netze in der Vergangenheit nur eine andere beteiligte Ressource, die benutzt und verwaltet zu werden braucht. Aus der Applikationsperspektive wurden die passive Netze als ein "*black box*" ausgesehen. Applikation weiß leider nicht,was in diesem "*black box*" zur Zeit passiert, und weiß nur *Input* sogar *Output* von diesem "*black box*". Der Anfang des Internetentwurfs wurde mit der *End-to-End* Systemkonzept konstruiert [Kust00]. Die inneren Knoten wie *Routers,Bridges* haben Funktionen nicht mehr als Speichern und Paketsvorwärtssenden. Aber als Internet sich vergrösst, haben mehr und mehr Benutzer solches Problem getroffen, sie und ein paar Type von den Applikationen haben die Ausnutzung der verteilten Netzressourcen höher und komplex gemacht. Außerdem brauchen manche Applikationen die Dienste, die best unterstützt oder Informationsbenutzung nur inner der Netze erhöht werden können. Wenn das Netz die mit den Applikationen verbundene Information dauerhaft benutzt, z.B. wie Zeit und Position der Stauung, *hot-spot* Punkte im Netz, lokale Position der verloren Pakete, usw. Es wird signifikant erfordert, den Dienst von der Applikation gesehen werden zu können. Wenn die Applikationen die Informationen, die von den Netzen gebraucht werden, weiter geben können, dann wird es idealer Dienst der Netze. So kann man sich mal überlegen, wie man einen neuen Dienst im aktiven Netz einführen kann. Zuerst sollte ein Dienstsystem alle Netzinformationen und Applikationsinformationen in einem Dienstsystem anzeichnen. Dann könnte dieses Dienstsystem die Benutzer oder die Applikationen freie *Requests* im aktiven Netz zur Verfügung stellen und den neuen Dienst für das Netz als Benutzer- oder Applikationsgebrauch anpassbar und interaktiv geben lassen. Man kann so vorstellen, vorher kann man nur wissen, *Input* und *Output* von dem "*black box*". Aber jetzt wird ein Dienstsystem mit allen Informationen von der Benutzerseite, Applikationsseite und Netzseite erzeugt. Dannach werden eigene *Requests* von den Benutzern inner *Inputstrom* ins "*black box*" gesendet. Im "*black box*" spielt der aktive Dienst zu seiner Rolle als ein intelligenter Spion, er überwacht und weiß es schon, z.B. welche Paket ist gerade verloren, wo steht das verlorene Paket, auf welchem *Router* gibt es eine Stauung, wie lange dauert die Stauung, usw.. Zwar noch kann der aktive Dienst als ein sorgfältiger *Manager* spielen, er besorgt sich dafür, wann *Requests* von der Benutzerseite berücksichtigt werden, was nach den *Requests* im Netz funtionieren soll, und wie es funktioniert. Das aktive Netz

kann seine Ressourcen gut verwalten, um gute Ausführung in seinen Diensten zu ermöglichen. Hier kann das aktive Netz nicht nur speichern und Pakete vorwärts übertragen, sonder auch berechnen, nach den Benutzern oder Applikationen übertragen und den beiden Gastfreundschaft geben.Das ist der Zweck der Einführung des aktiven Dienstes in den aktiven Netzen. Tatsichtlich sollte die Einfürhung des aktiven Dienstes so sein, immer mehr und neue Funktionalitäten ins aktive Netz anzubieten und neue Forderungen von den Benutzern oder den Applikationen schnell zu leisten. Auf der anderen Seite des Netzes sollte es neue Dienste für die Benutzer oder die Applikationen zur Verfügung stellen. Früher hat man leider nur kleine Chance, solche "intelligente" Dienste im Netz zu einführen, weil das Netz selbst passiv ist. Das Netz funktionierte nur als "*postman*". Es transportierte Bits von einem Endsystem zu anderen ohne zu wissen, was er mitbrachte. Es war sehr schwer, sich die Einführung des Dienstes zu entwickeln. Aber jetzt kann das Problem mit moderner aktiven Netztechnologie viel besser gelöst. Die Einführung des neuen Dienst in den aktiven Netzen kommt von zwei Seite, den vorhandenen Dienst in den passiven Netzen flexible zu verbessern und neuen Dienst in aktiven Netzen mit neuen Funktionalitäten zu leiten. So muss man zuerst einige Masse Ahnung über die aktiven Netze haben. Die wichtigen Eigenschaften der aktiven Netze und was der aktive Knoten ist, der große Rolle für die aktiven Dienste spielt, sollten gewusst werden. In diesem Artikel wird so strukturiert:am Anfang wird es im ersten Abschnitt vorgestellt, was das aktive Netz ist, die Eigenschaften de aktiven Netzes, Vor- und Nachteile des aktiven Netzes, den aktiven Knoten, das aktive Knotenoperationssystem und das Ziel der Anwendung des aktiven Netzes;im zweiten Abschnitt kommt die Dienste im aktiven Netz, zuerst wird es vorgestellt, was der Dienst eigentlich ist, im Vergleich zum Komponenten, die klassische Definitionen von der Komponente und dem Dienst. Dannach werden die Methoden und die Mechanismen zur Erzeugung der aktiven Dienste vorgestellt. Durch zwei Bespiele, *Web Caching*[J.We99] und *Demand Inquiring Service*[Kust00], wird es in diesem Bereich vertiefert. *Web Caching* verbessert einen in passiven Netzen schon vorhandenen Dienst in aktiven Netzen und *DIS* führt einen neuen Dienst in aktiven Netz ein;zum Schluss dieses Artikels wird die Zukunft der Forschung an den aktiven Netzdienste kurz vorgeschaut.

2 Die aktiven Netze(*Active Networks*)

Ähnlich meiste Netze besteht ein *Active Networks* basiertes Netz aus einer interzusammengehängten Gruppe von Knoten, die normale *runtime* ausführt. Die Knoten könnten über lokale- oder weitere Orte und durch Punkt-zu-Punkt(*point-to-point*) oder aufteilende Mediumkannele zusammengehängt werden. Im diesem Abschnitt wird ein gründlicher Eindruck auf die aktiven Netze erst gehabt.

2.1 Grundlage der aktiven Netze

Traditionale Datennetze transportieren Bits von einem Endsystem nach anderen aber passiv. Es wäre ideal, wenn Benutzerdaten undurchsichtig übertragen werden. d.h. die Netze sind unempfindlich gegen getragene Bits und die Bits werden ohne Modifikation zwischen Endsysteme übertragen. Im Vergleich zur traditionalen Phase erlaubt die aktiven Netze die Netze kundenorientiertes Rechnen der Benutzerdaten auszuführen. z.B. Ein Benutzer von einem aktiven Netz kann ein kundenorientiertes Kompressionsprogramm an einem Knoten ins Netz schicken und fordert den Knoten das Programm auszuführen beim Verlauf deren Pakete auf.

Es gibt zwei Phase die Netze zum "aktivieren"[TSSJ$^+$]:

1. *Switches*/Schalter führt Rechnen der Benutzerdaten fließend durch denselben aus.Diese sogenannte "*the programmable switch approach*" Phase hat existierte Paket/*Cell* Format

und versorgt einen diskreten Mechanismus, der die Unterladenphase der Programme unterstützt. Trennung der Programminjektion vom Verlauf der Nachrichten die Selektionen von den Programmen im Gegensatz zu den individullen Endbenutzer machen. Unabhängig kann die Programminjektion auch ins Netz getragen werden, dadurch dass der Verlauf am Knoten zu den Benutzer- und Applikationsspezifikationen geschnitten wird.

2. "*Capsule approach*" Phase ersetzt passive Pakete im heutigen Netzarchitektur durch die aktiven miniaturlichen Programme, die in der Übertragungsframe entkapselt werden und an jedem Knoten entlang ihren Weg ausführen. Der in der Architekturperspektive von den passiven Paketen zu den aktiven *Capsules* Wechsel adressiert gleichzeitig erst die beschriebene aktive *Capsule*. Durch die Miniprogrammform könnten Benutzerdaten in diesen *Capsules* eingepackt werden. Beispielsweise werden die *Contents* einer Webseite durch eine von Postscript codierte Fragment eingepackt. Weiterhin kann *Capsule pre-defined* Programm abrufen oder neues Programm in den Netzknoten planen. Abb.1 zeigt normale *Capsule* Format[WeLG98]. .

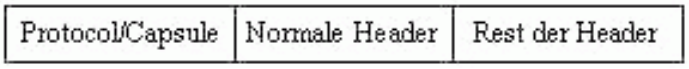

Abbildung 1: *Capsule* Format

Der wichtigste Aspekt der Forschung im Bereich der aktiven Netze liegt in Technologie "*Push*" und Benutzer "*Pull*"[TSSJ$^+$]. Technologie "*Push*" ist definitiv Auftauchen der aktiven Technologien, beispielsweise Kompilierung und Interpretation, Unterstützung der Entkapselung, Übertragung, Interposition, sichere und effiziente Ausführung der Programmsfragmenten. Solche heutigen aktiven Technologien werden zwischen individuelle Endsysteme und die über der *end-to-end* Netze Schicht verwendet.z.B. Programmsfragmenten wie *Java-Applets* können zwischen den *Web Server* und den *Client* ausgetauscht werden. Die Denkmethode in den aktiven Netzen wird im Vergleich zu den passiven Netzen von "in" den Netzen nach "innerhalb" den Netzen gewechselt, dadurch dass die aktiven Netze die obengenannten aktiven Technologien zum Nutzen beeinflussen und erweitern. Auf der anderen Seite kommt Benutzer "*Pull*" Phase aus einer sortierten Menge von *Firewalls,WebProxies,multicast routers,mobile Proxies,video gateways*, usw., die das von Benutzer getriebenes Rechnen an den Knoten "zwischen" der Netze ausführen können. In vielen Fällen werden diese Dienste an den Knoten implementiert, wie *Firewalls*, die die Fassade der *Routers* anwenden aber die konventionalen architekturellen *Guidelines* den übersteigenden applikationsspezifischen Forschritt ausführen. Der Zweck ist bloß eine Ersetzung der ein paar besonderen Methoden zum Netzbasierten Rechnen mit einer allgemeinen Fähigkeit, die Benutzer ihre eigene Netze zu programmieren erlaubt. Das bedeutet, die Netze sollten nicht mehr über den Benutzer sondern Benutzerorientiert sein.

2.2 Das aktive Knotenoperationssystem

In den aktiven Netzen spielt der aktive Knoten sehr wichtige Rolle. Um sichere Ausführung der neuen Dienstprogramme inner den Netzen zu bestimmten erstellen die aktiven Netze eines aktives Knotenoperationssystems[J.We99]. Es wird spezial dargestellt, dass das obene System auf der Höhe von einem konventionalen Operationssystem wie Unix aufgebaut wird, weil seine Mechanismen effizient für die bei der Vorwärtsrate von der *Capsule* vervollständigenden *Tasks* ausführen müssen. Weiter kann man sich so vorstellen, das Knotenoperationssystem beeinflusst Mechanismen, dass die die vorwärts Routinen als unvertrauliche Code behandeln,

wegen der Skale der Internet und seiner omposition als eine Kollektion von den autonomen Systemen. Erstens, das aktive Knotenoperationssystem stellt Protektionsmechanismen zur

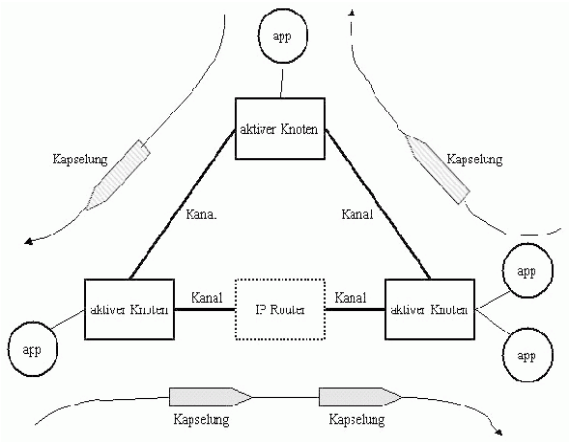

Abbildung 2: *Eintities* in einem aktiven Netz

Verfügung. Die isolieren verschiedene Dienstcode und verhüten die vom verderben der Netze. In den aktiven Netzen wird sich die Isolationsform *Fingerprintbased* Protektionsmodel entwickelt[J.We99]. d.h. es ist unmöglich, z.B., einen Dienst zu konstruieren, der zu einem unabhängigen Benutzer gehörte Pakete sucht und ablegt.Weiterhin, das aktive Knotenoperationssystem stellt Ressourcenmechanismen zur Verfügung. Die Operation kann auf einen Dienst mit anderen grundlos zu mischen verhindern[J.We99]. In den aktiven Netzen werden Probleme durch die vor freier Ausführung von einer vertraulichen Authorität lizenzierten Dienstcodebenötigung gelöst. Der Mechanismus unterscheidet ich von dem Rest des Knotenopeartionssystems, der an unvertrauliche Code arbeiten kann. Abb.2 zeigt *Entities* in einem aktiven Netz mit einem Knotenoperationssystem und Abb.3 zeigt *Demand Loading of Protocol Code*. .

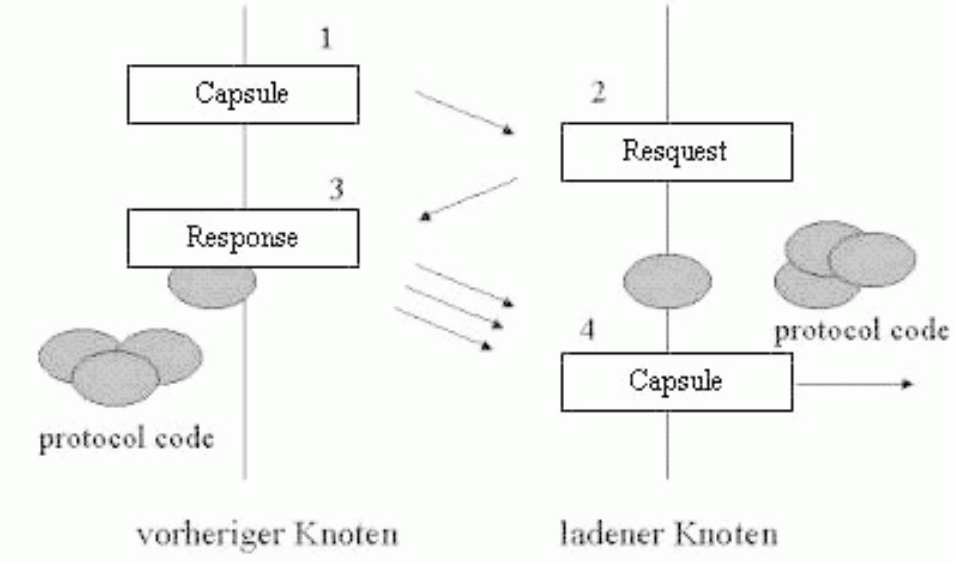

Abbildung 3: *Demand Loading of Protocol Code*

2.3 Vor- und Nachteile des aktiven Netz

Es ist schon bekannt gewesen, was das aktive Netz, der aktive Knoten und das aktive Knotenoperationssystem ist. Aber warum will man das aktive Netz erstellen? In Vergleich zu dem passiven Netz hat das aktive Netz folgende Vor- und Nachteile. Vorteile:hauptsächlich gibt es drei Vroteile zum Aufbau des aktiven Netzarchitekturs mit Austausch der aktiven Programme statt den passiven Pakete:

- Austausch der Code beschafft eine Grundlage für adaptive *Protocols* und kann mehr Interaktionen als Austausch der *fixed data* Formaten.
- *Capsules* versorgen eine Bedeutung der Implementierung der finen geammelten applikationsspezifischen Funktionen an den strategischen Punkten zwischen den Netzen.
- Die Programmabstaktion beschafft eine mächtige Plattform für das von den Benutzer getriebenes Kundenorientieren von der Infrastruktur und lässt neue Dienste auf einen schnelleren Schritt entfalten als die von Verkäufer getriebene Standardisierungsphase zu erhalten.

Aber die aktiven Netze haben auch Nachteile.Mehr Komplexität als in den traditionalen passiven Netzen wird steigend beträchtliche Sicherheit immer aufgefordert. Und die Standardisierung und die geforderte manuelle rückwärtskompatible Stellung sind sich leider ziemlich schwer und lang zu entwickeln. Die Forschungskosten sind zur Zeit bestimmt mehrfach als in den passiven Netzen.

2.4 Ziele der aktiven Netze

Was ist der Zweck der Entwicklung der aktiven Netze eigentlich?Warum wird Netzprotocols im Aufbau der aktiven Netze erstellt?Es gibt doch drei Gründen. Alle schreiben mehr flexble Forme der Erstellung als die heute benutzten. Im allgemeinen sollen die Netze die folgende Sache unterstützen:

- gleichzeitige Nutzungen einer Abwechselung von den verschiedenen *Protocols* auf den verschiedenen Netzschichten;
- Konstruktion und Anwendung neuer *Protocols* lieber durch gemeinsame Bestimmung in interessierten *Parties* als Verlangen neue *Protocols* in einer zentralisierten Weise registeriert zu werden;
- Die dynamische Erzeugung der neuen *Protocols* ohne *Reconfiguration* besonders als die Skale der Inkrement in den Netzen.

In diesem Abschnitt wird es angeschaut, die Definition der aktiven Netze sogar des aktiven Knoten und des aktiven Knotenoperationssystems, die Vor- und Nachteile der aktiven Netze und die Ziele der aktiven Netze.Der Spielregel in den aktiven Netzen ist schon bekannt. Jeder Benutzer sendet seine Programme ins aktive Netz und der aktive Dienst danach funktioniert dann verschiedenen Applikationen am aktiven Knoten inner dem Netz. Im folgenden Abschnitt werden die Dienste in den aktiven Netzen vorgestellt.

3 Die Dienste in den aktiven Netzen

Das Tempo der Erneuerung in den Netzapplikationen ist gnadenlos.Neue Applikationen gehen weiter zum schnellen Auftauchen und werden von den neuen Netzdiensten beigetragen, die sich an ihren selben Nutzenmodel anpassen. Es ist möglich neue Netzdienste an Endsysteme zu entfalten und die an den Knoten inner den Netzen zu implementieren oder oft bessere Funktionalitäten und Performance auf der Netzschicht anzubieten. Zu diesem Thema sollte man zuerst wissen, was Dienst eigentlich ist.

3.1 Die klassische Definition des Dienstes

Um die klassische Definition des Dienstes zu wissen braucht man zuerst kennen, was Komponente ist. Eine Komponente ist ein *Eintitiy*, das ein von den Komponentenentwicklern definiertes *well-defined Interface* und Verhalten[Grou98]. Komponenten formieren grundsätzlichen Aufbau des Dienstentwurfes. Eine Komponente soll nicht nur eine Spezifikation sndern auch eine Implementation haben. Die Spezifikation schließt die bzgl. auf die Komponenten Methode, die Komponenten auszuführende Schnittstelle und eineDeskription des Verhaltens von den Komponenten ein. Die Implementation realisiert in der Spezifikation gegebene Schnittstelle und Verhalten. Eine Komponente könnte Regeln haben, die basiert auf Semantik diktieren, wie die benutzt werden könnte. Diese klassische Definition der Komponente lässt einen weiten Bereich von den Optionen für Komponenten von einfacher Programmsprachenfeststellung bis zu komplexen Verhalten zu. Ein Dienst ist ein *Entity*, das ein von den Dienstentwicklern definiertes *well-defined Interface* und Verhalten hat[Grou98]. Ein wichtiger Unterschied zwischen einen Dienst und eine Komponente liegt da, ein Dienst ist “sichtbar” für den Benutzer. Manche Komponenten könnten auch Dienste identisch sein. Typisch werden die Dienste durch Einsatz(*Putting*) der multiplen Komponenten miteinander erzeugt, dadurch dass ein sequentieller Kontrollmechanismus von einer kompositionalen Methode ist Syntax und Sematik zur Diensterzeugung aus den Komponenten. Die enthält Spezifikation aber keine Implementation von den Komponenten.

3.2 Methode und Mechanismen

Nachdem die klassische Definition des Dienstes gewusst worden ist, hat man jetzt Lust dran, wie spielt der Dienst weiterhin in den aktiven Netzen. Nicht ähnlich IP ist der von den aktiven Netzen ausgeführte Netzdienst nicht fest sondern flexble. Netzdienst ist aktiv, wenn das Netz für die durch Daten- oder Paketkommunikation zwischen den Benutzern oder den Applikationen neue *Protocols* in die Netze vorzustellen. Manchmal ist es transparent und es ist manchmal nicht von dem Punkt der Benutzersicht. Von der Benutzerseite sind nur ein Teil der Netzdienste transparent und anderer Teil nicht. Von der Netzsicht gibt es ein paar von *Internet Engineering Task Force(IETF)* empfehlende Dienstmodele und Mechanismen[J.We99]: -*Integrated Services/Resource Reservation Protocol(RSVP) Model*; -*differentiated Services(DS) Model*; -*multi protocol label switching(MPLS)*; -*traffic engineering*; -*constraint-based routing (CBR)*; Die obengenannten Methoden oder Mechanismen versuchen die Dienste zu erzeugen, die Reservation oder Klassifikation der Dienste der Paketübertragung machen können statt beste Dienstleistung zu liefern. Aber alle Dienste von diesen Model oder Mechanismen sind unberührbar von der Benutzersicht. Die Netzdienste wird so vorgeschlagen, ihre Benutzer ihre eigene Befehle auf dem Rang der Zustimmung der normalen Dienstschicht anfordern können.

3.3 Beispiele

In diesem Abschnitt wird zwei Beispiele, *Web Caching*[J.We99] und *Demand Inquiring Service*[Kust00], vorgestellt. Und dadurch wird es erklärt, wie die Dienste in den aktiven Netzen eingeführt werden.

3.3.1 *Web Caching*[J.We99]

Web Servers benutzen *Cache* und *laden* verteilungsdienste.z.B. *Cisco's Cachedrector* Produkt reduziert den Betrag des weltweiten Handels durch Erstellung der Wiederholung der *Requests*, und *Cisco's Localdirector* Produkt reduziert die Konzentration auf Web Handel durch Verteilung der *Requests* über multiple Dienste. Bei der Erstellung der Pakete an den Knoten sind obene Produkte offensichtlich zu den Endsystemen und minimiert die Bandbreite und Latenz um Vergleich zum *Proxy Agent*. Auf dieser Welt inkrementiert Web Handel explosiv, dafür spielt *Caching* Mechanismus große Rolle. Sie haben drei Ziele: -verniedrigen Latenz von *Client Requests*; -verniedrigen die Loadzeit auf dem *Server*; -verniedrigen Benutzung der *Wide-area* Bandbreite; Es ist nötig Information zwischen *Caches* an verschiedenen Lokalen in den Netzen zu teilen, um *Caching* noch weiter zu entwickeln. Aber die stichwörtlichen Probleme sind: Wie kann ein Kopie einer Dokument durch Cachesuchen kokalisiert werden? Wie können die Dokumente nach *Caches* für weitere Absichte forpflanzt werden?Ja, es ist schon klar, die beiden Probleme können doch an Terme von den *Routing Web-Requests* sogar -*Responses* via ein beteiligtes Netz der *Caches* nachgedacht werden. Das stellt eine neue Frage vor uns, wie man überall *Cache Lookup* und Ersetzungpfad mit einem neuen Netzdienst, der *Routing* und *Caching* kombiniert, kontrollieren kann. So sollte man vielleicht den Dienst vorstellen und designieren. . Dienst Design Zwei Funktionalitäten müssen besonders

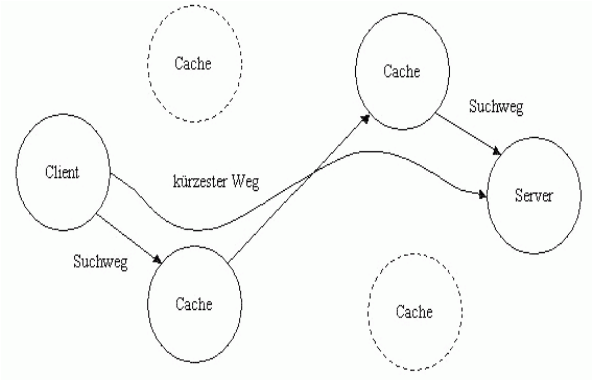

Abbildung 4: Komination *Web Cache Lookup* mit netzrouting

berücksichtigt werden, wenn ein *Caching* Dienst eingeführt wird. Es muss irgende Wege für *Web-Requests* zu lokalen Objekten in *Caches* geben, die einen kurzen Abstand von dem nach dem *Server* vorwärts laufenden Pfad sind. Auf der rückwärts Direktion muss es eine Bedeutung für *Web-Responses*, Objekte nach diesen gleichen *Caches* zu übertragen, geben, so dass sie möglich zufrieden mit weiteren zukünftigen *Requests* sind. So wenn man solche Mechanismen einführen möchte, dann sollten zwei Ziele getroffen werden. -Der Stoß in den Netzen muss minimal sein; -Der Stoß der Webzuverlässigkeit muss auch minimal sein; In den aktiven Netzen wird der Ergebnisdienst durch vier *co-operating capsule types* realisiert:

- *Bind Capsules* werden zwischen die *Caches* und ihre assoziierten aktiven Knoten ausgetaucht, um ihre Zustände zu synchronisierten.(Abb.5[J.We99])

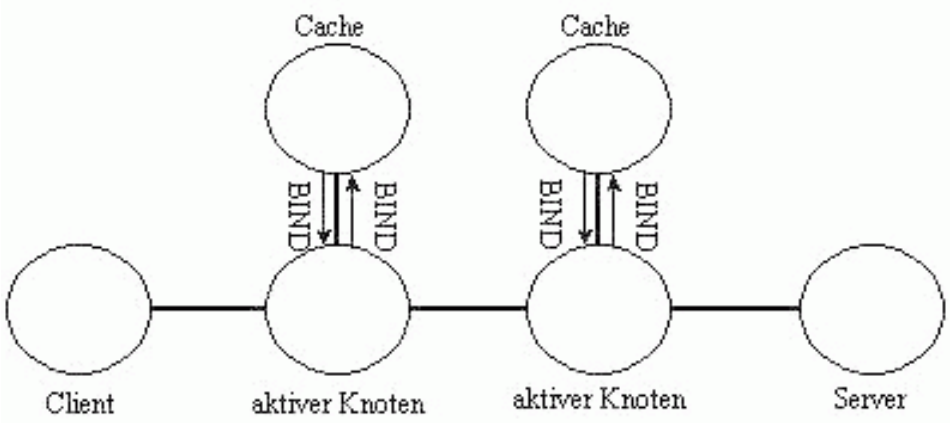

Abbildung 5: *Bind Capsules*

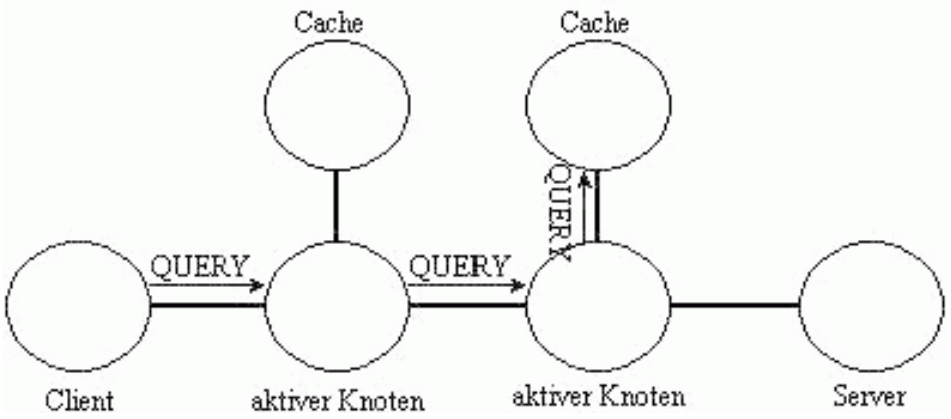

Abbildung 6: *Query Capsules*

- *Query Capsules* werden von *Client* nach dem *Server* gesendet. Entlang den Weg könnten sie *Caches* begegnen, die ein Kopie der *Requested* möglichst verschiedenen Dokumenten hat.(Abb.6[J.We99])

- *Redirect Capsules* initialieren eine Verbindung zwischen den *Server*(oder ein *upstream Cahce* und ein *downstream Cache*(oder *Client*),um eine Dokument un *redirecting* Subsequenz-*requests* zu übertragen.(Abb.7[J.We99])

- *Activate Capsules* terminieren eine Verbindung zwischen ein *downstream Cache*(oder *Client*) und *Server*(oder ein *upstream Cache*), gleich wenn die Übertragung einer Dokument vervollständigt und Redirectionzustand veröffentlicht werden.(Abb.8[J.We99])

Außerdem werden reguläre Pakete, die nicht spezielle Netzbehandlung fordern, zwischen den *downstream Cache* und *Client* ausgetauscht, *Web* Dokument selbst übertragt zu werden.

Im Allgemeinen können die heutige *Web Caches* die erzeugenden Daten nicht dynamisch speichern, weil *Responses* jede Zeit verschieden sein könnten. Der obenvorgestellte Dienst hat ein paar attraktiven Eigenschaften, aber die wichtige lautet: Abweichungen können konstruiert werden, die gleiche *Caching* Infrastruktur in den verschiedenen Zwecken zu benutzen. Der Author von *Web Caching* stellt noch zwei hypothetische Abweichungen mit verchiedenen Suchenphase und Ersetzungsüberwachung dar. Eine Abweichung extendiert Clientfähigkeiten. Der zweite Abweichung verbessert *Server*. Natürlich gibt es noch viele Abweichungsmöglichkeiten[J.We99].In den passiven Netzen gibt es *Cache*-Technologie schon.

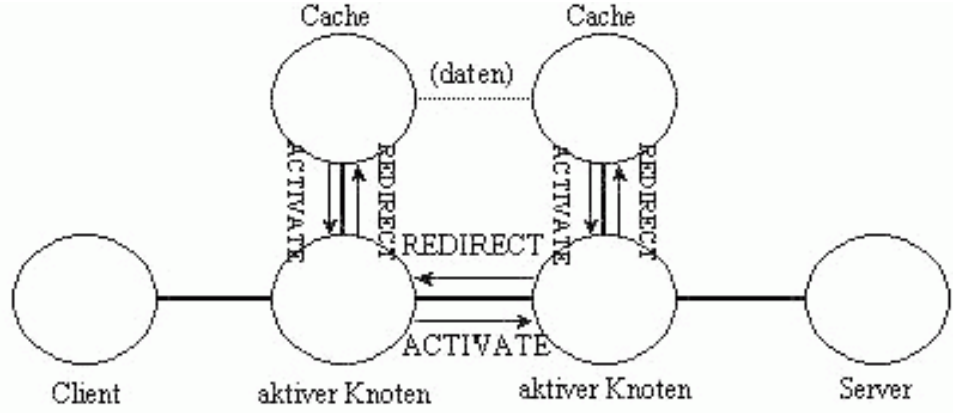

Abbildung 7: *Path of Capsules that load a downstream Cache*

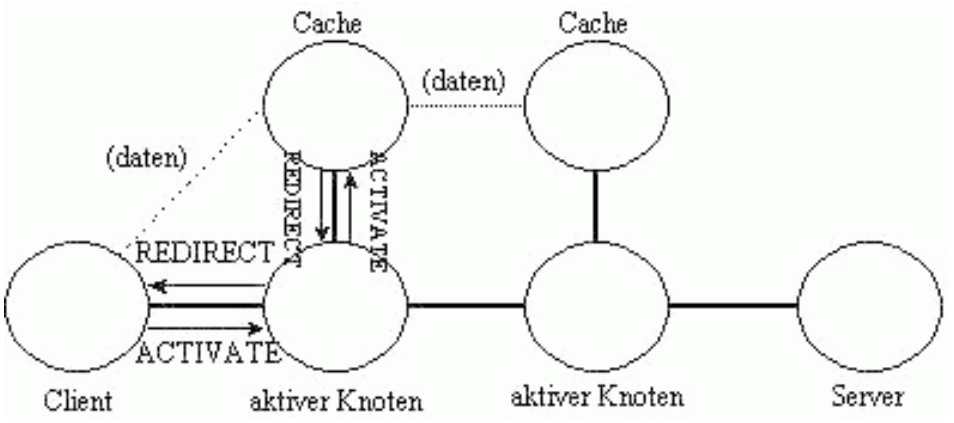

Abbildung 8: *Path of Capsules that return a document to the client*

Durch *Web Caching* kann es verbessert werden und die Koste wird abnehmen.Zwar noch kann es flexible sein.

3.3.2 *Demand Inquiring Service*[Kust00]

Ein Netzdienst basiert auf die Applikationsforderungen und Netzzustände. DIS kann individuelle Applikationsforderungen empfangen und Netzzuständensinformation berücksichtigen,um die beste Vorstellung der Netzressourcenmanagement zu erreichen und Netzinformation den Dienst zu den Applikationen zu beraten.

Struktur von DIS Wie sieht alle Elemente sogar ihre relativen Positionen im DIS System aus?

Netzmodel DIS benutzt *Client-Server* Model. Es gibt zwei Teile, *Client*-Seite und *Server*-Seite. Im *Client*-Teil gibt es die Applikation und DIS *Client*. Die Applikation verbindet DIS *Server* durch DIS *Client*. DIS läuft über andere *Protocols* wie RSVP, differentialen Dienst, *Routing Protocols* usw.. Und diese relative Position kann auf der Internetnetzsicht angeschaut werden. d.h. DIS läuft nicht getrennt bei der Diensteinführung sondern hat Korrelation mit anderen *Protocols*, die Applikationsforderungen auszuführen. Abb.9 zeigt die Relationspositionen der Elemente von DIS im Netz[Kust00]. DIS *Client* Die Applikation benutzt DIS *Client* bei der Verbindung mit DIS *Server*.Man kann so sagen, DIS *Client* wird als eine Bibliothek implementiert. DIS *Client* Bibliothek spielt Rolle als eine Schnittstelle zwischen Applikationen und DIS *Server*. Ein DIS *Client* bauet eine Verbindung mit den nächsten DIS *Server* auf und liefert die Applikationsforderung zu den DIS *Server*. Um diese Verbindung mit den nächsten DIS *Server* aufzubauen, erzeugt DIS *Client* alle Pakete, die von DIS besorgt werden und

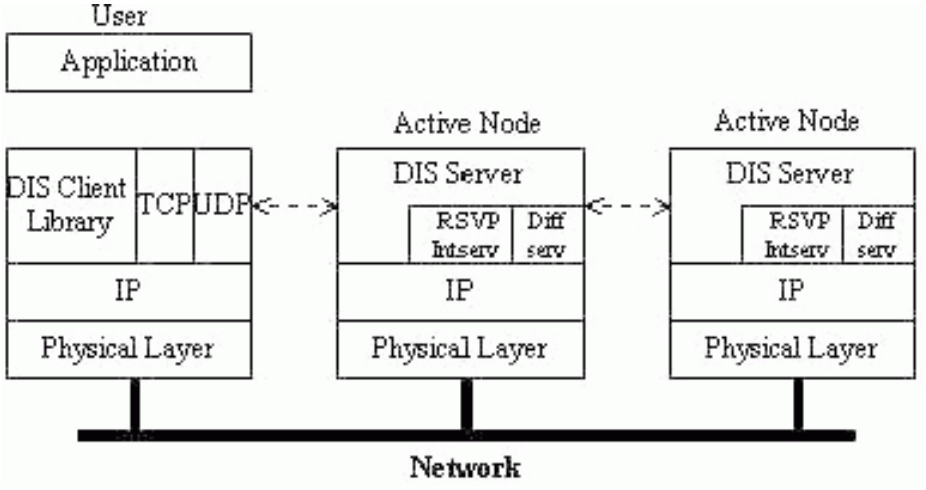

Abbildung 9: die Relationspositionen der Elemente von DIS im Netz

als DIS Pakete identifiziert.(Abb.10[Kust00]) Wenn DIS *Client Requests* an den DIS *Server*

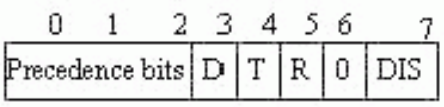

Abbildung 10: Identifizieren der DIS Paket von dem *IPv4Header*

zuschickt, als ein ID von *Request*, eine unique Nummer,eine Kombination einer Portnummer, was für ein Übertragungsprotocol, IP Adresse der Sender und Empfänger werden benutzt.Die Applikationsforderung kann als ein Programm oder eine Sequenz der Dienstparameterrepräsentiert werden. Bei der DIS Phase wird *Request* als QoS(*Quality of Services*), CoS(*Class of Services*) und CtS(*Condition to Serve*) klassifiziert. QoS enthält Parameter von Bandbreite, *Delay,Jitter*,Fehlerrate,Schedule,*Throughput*,usw..CoS ist eine Definition der Dienstklasse, wie hoch, medium, niedrig. CtS bezeichnet Parameter oder Programm, die den Zustand für den *Request*dienst repräsentiert. Einfaches Beispielprogramm für CtS: DIS *Server* Wie oben-

```
if (delay <= 50) then
        grand access to the resources;
else if (error rate < 10) then
        access to the resources in half level;
else if (server down) then
        wait 10;
else
        cancel request;
```

Abbildung 11: Beispielprogramm

gesprochen läuft DIS *Server* an den aktiven Knoten. Der aktive Knoten wird schon im ersten Abschnitt schon berücksichtigt, wie *Router* oder *Gateway*, der nicht nur speichern und Pakete vorwärts übertragen sondern auch selbst berechnen kann, wenn es sinnvoll nötig ist. Das ist die wichtige Konzept von den aktiven Netztechnologien.(Abb.12[Kust00]) DIS *Server* im aktiven Knoten ist ein Teil Netzdienst zu kontrollieren, die zu den Benutzern und Applikationen erzeugt werden. Dieses Teil empfängt ein *Request* von der Applikation in der Form von einem

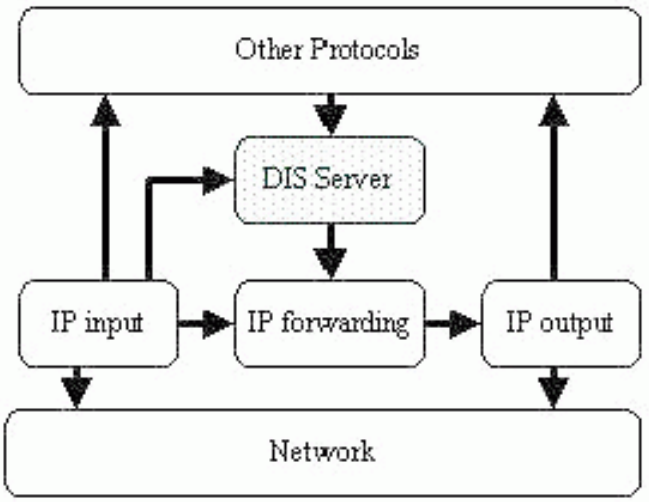

Abbildung 12: Fluß der Pakete an den aktiven Knoten mit DIS

Sourceprogramm oder Parameter. Zur Zustandserkennung der Netze alle Zeit tauscht jeder aktive Knoten Information periodisch zwischen den anliegenden Knoten, dabei sammelt der aktive Knoten Zustandsinformation in ihm seblst. Und von der Zustandsinformation dieser Netze kann sich DIS im aktiven Knoten entscheiden, ob die Netze dieses *Request* ausführen wollte oder nicht. DIS *Server* besteht aus vier Modulen(Abb.13[Kust00]):

1. *Reception Module*:der empfängt *Requests* von der *Client* Seite und extraktiert *Request* ID und identifiziert sie. Dann überträgt *Reception Module Request* nach zweitem Modul, sogenanntem *Request Analyzer Module. Reception Module* spielt eine Rolle, die Ergebnisse der *Dicision Module* zu empfangen und Applikation die Information weiterzugeben. Dieser Modul verwaltet das *Request*, die Status, und den Report zur *Protocol-correlation*,wie RIP,OSPF,RSVP,differentiale Dienste, MPLS, usw..

2. *Request Analyzer Module*:Hier wird *Request* dadurch analysiert. Erst analysiert *Request Analyzer Module* die Form von einem *Request*, ob es ein Programm oder eine Dienstparametersequenz ist. Dann wenn *Requestform* ein Sourceprogramm ist, wird Programmgültigkeit geprüft. d.h. Wie lange es diesem Programm noch lebendig dauert. Wenn ein Fehler dabei auftaucht, sendet Fehlermeldung an den *Client*, der so angefordert hat. Wenn ein Programm eine Dienstparametersequenz ist, dann analysiert alle Dienstparameter, die erzeugt werden können.

3. *Decision Module*:Dieser Modul basiert auf der Ergebnisanalyse und der durch *Monitoring Module* gestellten Information. *Decision Module* liefert Möglichkeit aus, die *Request*ergebnis zu versuchen. Dann trägt die Information, um die besten Ausführungsmöglichkeiten zu versammeln. Diese Information kann als eine Zeitreschedulsergebnis oder neue Dienstparameter oder nutzbares *Protocol* oder die Netzzustandsinformationen geformt werden.So verfasst sich alle zu *Client* gestellten Informationen.

4. *Monitoring Module*:Dieser Modul hat zwei Funktionen. Die erste ist lokalen Zustand wie Stauung, Paketfehler und andere Parameter darzustellen. Die zweite Funktion ist Information mit den anliegenden Knoten auszutauschen. Dabei gibt der Modul lokale Information und empfängt Zustandsinformation über anderen anliegenden Knoten. Der Modul fordert also die Möglichkeit, der anliegende Knoten die sich Benutzer*Request* beteiligten Ressourcen zu teilen.Dann kann dieser anliegende Knoten gleiche Sache nach seinem anliegenden Knoten weiter machen.

Im Vergleich zu passiven Netz macht die Einführung des DIS-Dienstes das Netz flexble. Das ist neue Funktionalität nur in aktiven Netzen.

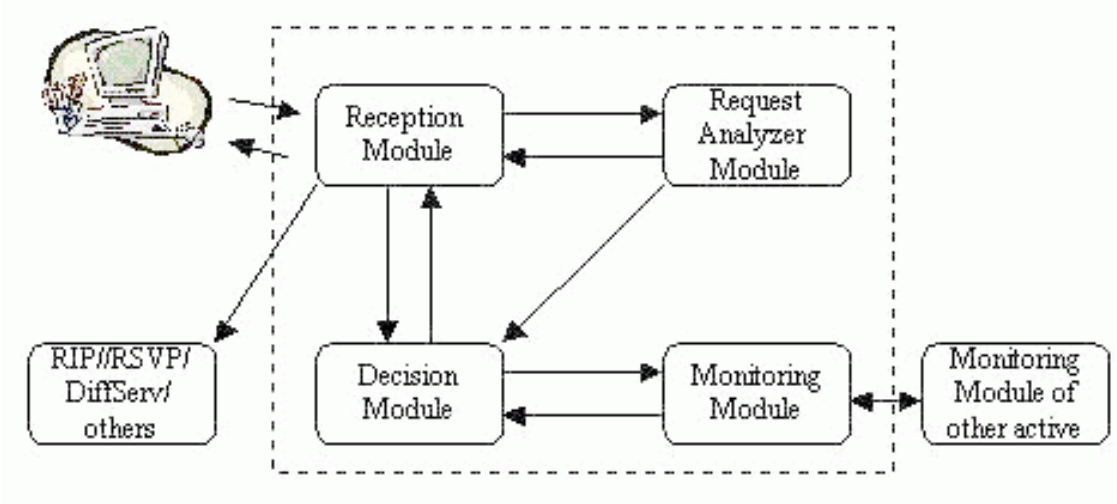

Abbildung 13: Verbindung zwischen Modules von DIS *Server* an den aktiven Knoten

4 Zusammenfassung und Ausblick

Es ist ziemlich schwer, einen neuen Dienst ins Netz einzuführen. Das Netz unterstützt die Einführung des neuen Dienstes.Aber der Dienst erwartet balancierte Flexibilität, d.h. Architektur des Netzes muss ausdruckvoll für die Anpassung an den neuen Diensten aber restriktiv genug um die Performanceziele und die Sicherheits*properties* getroffen werden zu erlauben. In diesem Artikel wird es durch zwei Beispiele zu erklären versucht, wie man einen neuen Dienst ins aktive Netz einführen kann. Die kombinierenden komplexen Beziehungen zwischen *Client,Server* und Applikation müssen zuerst berücksichtigt werden, um ersten Eindruck der Dienststruktur zu haben. Dann werden alle Aufgaben an den aktiven Knoten inner den Netzen, an *Client* und an den Applikationen verteilt. Nach dem Spielregel des Dienstes kann das aktive Netz selbst wissen, was es gerade gemacht hat und weiter machen soll. Es belohnt sich aber neue Dienste in aktiven Netzen erbracht zu werden. Die heutige allgemeine Richtung der Forschung an aktiven Netzdiesten geht nicht nur hin, wie man gute Dienste einführen kann, sondern auch wie man die Qualitäten der aktiven Netzdienste kontrollieren und verbessern. In der Zukunft sollte das Netz mehr anpassbar an der Erforderung vom unbegrenzten Bereich der Benutzerbefehle sein. Es wird unbedingt sein, die Perspektiven von der Telekommunikation und Rechergruppen in neuen dynamichen programmierbaren Netzarchitekturen, die für schnelle Diensteinführung und Ressourcenverwaltung durch eine Kombination von den netzbewussten Applikationen und den applikationenbewussten Netzen unterstützen, in Übereinstimmung zu bringen. Die programmierbaren Netze könnten die Dienste zu den besonderen Benutzern erzeugen, wie sie dieselbst geordnet haben.

Literatur

[Grou98] AN Composiable Services Working Group. *Composable Services for Active Networks*. September 1998.

[J.We99] David J.Wetherall. *Case Study II Web Caching*. 1999.

[Kust00] Hiroshi Yasuda Kustarto Widoyo, Terumasa Aoki. *Demand Inquiring Service:A Network Service Based on Application Demands an Network Condition*. 2000.

[TSSJ+] David L. Tennenhouse, Johathan M. Smith, W.David Sincoskie, David J.Wetherall und Gary J. Minden. *A Survey of Active Network Research*.

[WeLG98] David Wetherall, Ulana Legedza und John Guttag. *Introducing New Internet Services:Why and How*. 1998.

Abbildungsverzeichnis

JMF zur Entwicklung multimedialer Anwendungen

Nady Habashy

Kurzfassung

Das Java Media Framework (JMF) ist eine API (Application Programming Interface) für die Integration von zeitabhängigen Medien in Java-Applets/-Applikationen. JMF unterstützt die Erfassung, die Verarbeitung und die Präsentation von zeitabhängiger Mediendaten. Ein JMF *Player* präsentiert die Mediendaten aus einer Datenquelle. Als ein spezieller *Player* ermöglicht der *Processor* die Steuerung der Verarbeitung der Medienströme. Der *Processor* benutzt Standard-JMF-Plug-Ins oder angepasste Plug-Ins für die Verarbeitungsvorgänge wie das Demultiplexen der Eingabeströme, das Anwenden von Effektalgorithmen auf einzelnen Spuren, die Umkodierung der Mediendaten, und das Multiplexen. Das JMF Ereignismodell ermöglicht JMF Objekten, über den aktuellen Stand der Verarbeitung zu berichten bzw. auf die Ereignisse zur Laufzeit zu reagieren.

1 Einleitung

Die Werte von Medien wie Ton und Bewegtbild verändern sich über die Zeit hinweg. Die Information steckt somit nicht nur in einem enizelnen Wert, sondern auch im Zeitpunkt ihres Auftetens. Die Semantik kann auch im Gradienten der relativen Änderung der einzelnen Werte oder des Kontinuums bestehen. Solche Medien nennt man allgemein zeitabhängeige Medien [Stei00]. Eine der wichtigsten Eigenschaften zeitabhängiger Medien ist, dass sie rechtzeitige Übertragung und Bearbeitung erfodern. Beginnt der Datenfluss, so sind zeitliche Fristen streng einzuhalten. Audio-/Video-Clips, MIDI-Sequenzen und Animationen sind bekannte Formen für zeitabhängige Medien. Das JMF unterstützt die meisten standard Kontenttypen wie AIFF, AU, AVI MIDI, MPEG, QuickTime, RMF und WAV. Mit JMF kann man Applets und Applikationen entwickeln, die zeitabhängige Medien präsentieren, erfassen, bearbeiten und speichern. Fortgeschrittene Entwickler können mit JMF angepasste Verarbeitung von Mediendaten ausführen und JMF so erweitern, dass auch andere Kontenttypen und Formate unterstützt werden.

2 JMF-Architektur

JMF sieht ein geläufiges Modell für die Speicherung, Bearbeitung und die Präsentation von zeitabhängigen Medien vor. Eine Datenquelle enthält den Medienstrom, ähnlich wie eine Videocassette einen Videofilm enthält. Eine Abspielfunkion (Player) liefert die Mechanismen für die Bearbeitung und Steuerung, ähnlich wie bei einem VCR (Video Cassette Recorder). Für das Spielen und Erfassen von Audio und Video benötigt JMF die passenden Eingabe- und Ausgabe-Geräte wie Mikrofone, Kameras, Lautsprecher und Monitore. JMF hat eine höhere und eine niedrigere API. Die höhere API ist für die Erfassung, Präsentation und Bearbeitung von zeitabhängigen Medien. Die niedrigere API unterstützt die Integration von angepassten Komponenten und Erweiterungen.

2.1 Das Zeitmodell

Klassen, die das JMF-Zeitmodell unterstützen, implementieren die *Clock*-Schnittstelle, so dass die Zeit für einen "einzelnen" Medienstrom verfolgt werden kann. Die *Clock*-Schnittstelle liefert die Grundfunktionen für das Timing und für die Synchronisation, die notwendig für die Steuerung der Präsentation zeitabhängiger Medien sind. Eine *Clock* hat eine regelmäßig tickende Zeitquelle, die *TimeBase*, die einer Quarzuhr ähnelt. Mit der *TimeBase* verfolgt die *Clock* den Verlauf der Zeit während der Präsentation. Die MediaTime der *Clock* ist die aktuelle Position in dem Medienstrom, die mit einer MediaTime gleich Null beginnt und mit der maximalen *MediaTime* des Stromes endet. Die Dauer des Medienstromes ist die verstrichene Zeit zwischen dem Beginn und Ende des Stromes.

Um die aktuelle *MediaTime* zu verfolgen, definiert die *Clock* eine Transformation auf die Zeit, die *TimeBase* hält, wie folgt:

$$mt = mst + (tbt - tbst) * rate$$

wobei,
- mt (media-time): die aktuelle Position in dem Medienstrom
- mst (media-start-time) : die Position in dem Medienstrom zu Begin der Präsentation,
- tbt (time-base-time) : die aktuelle Zeit, die *TimeBase* liefert,
- tbst (time-base-start-time) : die Zeit, die *TimeBase* zu Beginn der Präsentation liefert,
- rate : ist die Playback Rate.

Da *TimeBase* durch den Aufruf der Methode getTimeBase die time-base-time liefert, kann media-time mit der Transformation berechnet werden. Die Playbackrate, bestimmt, wie schnell die *Clock* in bezug zu ihrer *TimeBase* läuft. Ist die Playbackrate z.B. 3.0 , so bedeutet das ,dass die *Clock* drei mal so schnell läuft, wie ihre *TimeBase*. Ist die Playbackrate negativ, so bedeutet das, dass die *Clock* in die entgegengesetzte Richtung läuft, wie ihre *TimeBase*, um z.B. einen Medienstrom rückwärts abzuspielen. Die Rate 1.0 ist die normale Playbackrate für Medienströme.
Die Parameter time-base-start-time und media-start-time definieren einen gemeinsamen Zeitpunkt, an dem die *Clock* mit ihrer *TimeBase* synchron ist.
Wenn die Präsentation stopt, stopt die MediaTime, aber die *TimeBase* schreitet weiter fort. Startet die Präsentation neu, wird die media-start-time mit der time-base-start-time neu synchrinisiert. JMF behandelt die Zeit mit einer Präzision im Nanosekundenbereich.

2.2 Die Manager

Das Java Madia Framework besteht hauptsichlich aus Schnittstellen, die das Verhalten und die Interaktion der Objekte, die für das Erfassen, das Bearbeiten und das Präsentieren von zeitabhängige Medien benutzt werden, definieren. Um die Integration von neuen Implementierungen der Schnittstellen zu vereinfachen, verwendet JMF die Manager. Die Manager sind Vermittlerobjekte zwischen den Schnittstellen und existierenden Klassen. Dadurch können die Schnittstellen gleichermaßen angewendet werden, ob es sich dabei um vordefinierte oder angepaßte Implementierung handelt.

Das Java Media Framework benutzt vier Manager:

- Manager - zur Konstruktion von Player, Prozessoren, Datenquellen und Datensenken.

- PackageManager - hält ein Register für die Pakete, welche die JMF Klassen enthalten.
- CaptureDeviceManager - hält ein Register für alle vorhandenen Erfassungsgeräte.
- PlugInManager - hält ein Register für die verfügbaren Plug-Ins, wie Multiplexer, Demultiplexer, Codecs.

Programme, welche die JMF API benutzen, legen Player, Prozessoren, Datenquellen und Datensenken mit der *Create*-Methode des Managers an. Zur Erfassung von Mediendaten über Eingabegeräte, wird der CaptureDeviceManager benutzt, um verfügbare Geräte herauszufinden und auf ihre Informationen zu zugreifen.
Mit dem PlugInManager werden neue Plug-Ins registriert, um sie für Prozessoren, welche die Plug-In API unterstützen, verfügbar zu machen. Zur Steuerung der Prozesse, die auf die Daten ausgeführt werden, wird dann der PlugInManager über registrierte Plug-Ins konsultiert.
Dar eindeutige Package-Name angepasster Player, Prozessoren, Datenquellen oder Datensenken wird mit dem PackageManager registriert, um sie dann mit dem JMF benutzen zu können.

2.3 Das Ereignismodell

JMF benutzt einen strukturierten Mechanismus für die Ereignisnachrichten, um JMF-Programme mit Informationen über den aktuellen Zustand des Mediensystems zu liefern und ermöglicht JMF-Programmen, auf medienbezogene Fehler wie „out-of data" zu reagieren.
JMF-Objekte können über die aktuelle Situation berichten, indem sie ein MediaEvent senden.
MediaEvent ist die Basisklasse für Medien-Ereignisse und hat die Unterklassen ControllerEvent, DataSinkEvent, GainChangeEvent und RTPEvent.
JMF definiert für alle Objekttypen, die ein MediaEvent senden können, eine *Listener*-Schnittstelle. Um eine Mitteilung zu erhalten, wenn ein MediaEvent gesendet wird, muss eine Klasse die entsprechende *Listener*-Schnittstelle implementieren und sich bei dem Objekt, das die Ereignisse sendet durch den Aufruf von *addListener* registrieren lassen.

2.4 Das Datenmodell

Eine Datenquelle enthält den Ort der Medien sowie das Protokoll und die Software für die Übermittlung der Medien. Die JMF Medien-Player benutzen gewöhnlich Datenquellen, um den Transfer von Medien zu verwalten. Eine Datenquelle kann nicht wiederverwendet werden, um andere Medien zu übermitteln.
Eine Datenquelle ist durch eine URL oder JMF MediaLocator gekennzeichnet. Ein MediaLocator ist ähnlich wie eine URL und kann aus einer URL gebildet werden. Im Gegesatz zu URLs, kann ein MediaLocator gebildet werden, auch wenn das entprechende Protokoll auf dem System nicht installiert ist.
Eine Datenquelle verwaltet eine Menge von *SourceStream*-Objekten. Eine standard Datenquellen benutzt ein Byte-Array als Transfereinheit, während eine Buffer-Datenquellen ein Buffer-Objekt benutzt.

JMF Datenquellen können nach der Art, wie der Datentransfer eingeleitet wird, kategorisiert werden:

- *Pull-Datenquellen* - Der Client leitet den Datentransfer ein und steuert den Datenfluss. Protokolle, die für diesen Datentyp verwendet werden, sind HTTP und FILE. JMF definiert zwei Typen von Pull-Datenquellen: *PullDataSource* und *PullBufferDataSource*.

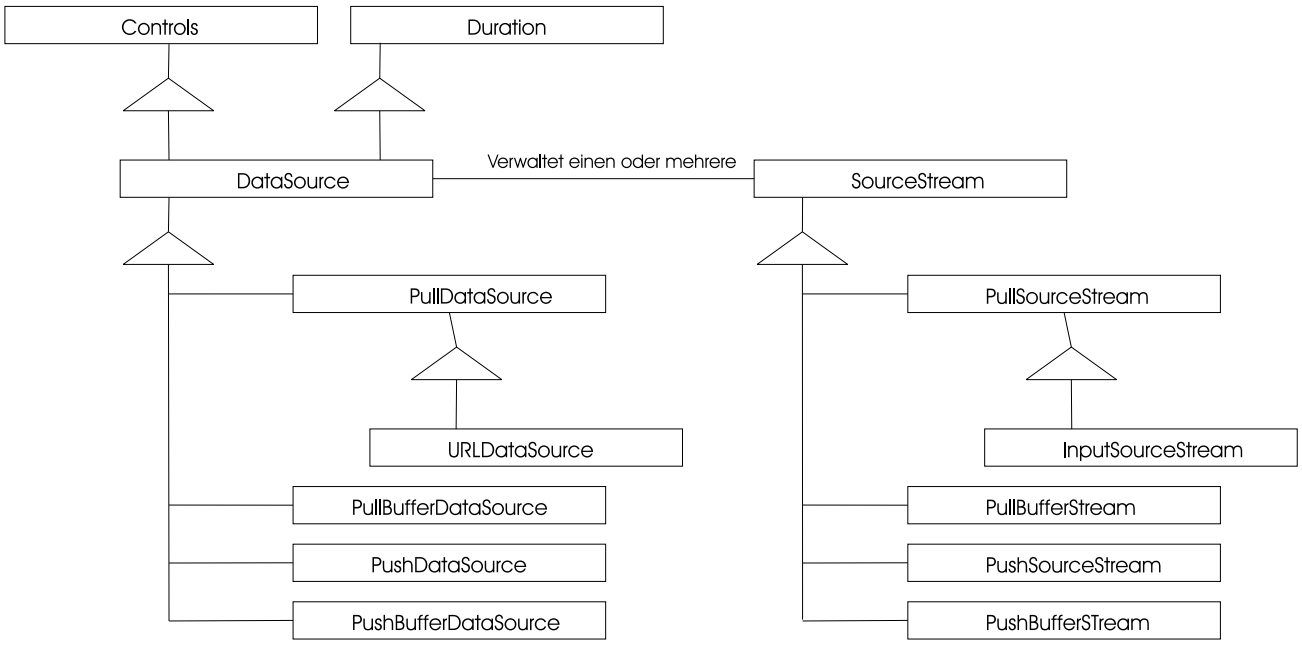

Abbildung 1: Das Datenmodell.

- *Push-Datenquellen* - Der Server leitet den Datetransfer ein und steuert den Datenfluss. *Broadcast*-Medien, *Multicast*-Medien und *Video-On-Demand* (VOD) sind Push-Datenquellen. *Real-time Transport Protocol* (RTP) ist das Protokoll für *Broadcast*-Daten. Für VOD ist *MediaBase* das Protokoll. JMF definiert zwei Typen von Push-Datenquellen: *PushDataSource* und *PushBufferDataSource*.

Inwieweit der Benutzer in die Steuerung eingreifen kann, hängt von dem Typ der Datenquelle ab. Eine MPEG-Datei kann z.B. neu positioniert werden und der Benutzer kann, wenn das Client-Programm es zulässt, das Video wiederholen oder eine neue Position im Video suchen. Bei *Broadcast*-Medien hat der Server die Steuerung und kann nicht vom Benutzer auf der Client-Seite repositioniert werden.
JMF hat zwei besondere Typen von Datenquellen, die *CloneableDataSource* und die *MergingDataSource*.
Eine *CloneableDataSource* wird benutzt, um Pull- oder Push-Datenquellen zu klonen. Mit dem Aufruf der Methode *createCloneableDataSource* des Managers mit der Datenquelle als Übergabeparameter wird eine *CloneableDataSource* erzeugt. Von da an sollen die *CloneableDataSource* und ihre Klone und nicht die originale Datenquelle benutzt werden.
Eine *MergingDataSource* wird benutzt, um die SourceStreams verschiedener Datenquellen in eine Datenquelle zu verknüpfen. Dies ermöglicht die Steuerung mehrere Datenquellen am einem Punkt. Funktionsaufrufe auf *MergingDataSource* werden an alle verknüpften Datenquellen weitergeleitet.
Eine *MergingDataSource* wird mit der Methode *createMergingDataSource* des Managers und ein Array von Datenquellen, die verknüpft werden sollen, als Parameter erzeugt. Die verknüpften Datenquellen müssen vom gleichen Typ sein. PullDataSource und PushDataSource können z.B. nicht verknüpft werden. Die Dauer einer *MergingDataSource* ist das Maximum der verknüpften Datenquellen.

2.5 Die Steuerung

Die JMF-*Control* unterstützt einen Mechanismus für das Schreiben und Lesen der Attribute eines Objekts. *Control* ermöglicht meistens die Steuerung der Attribute eines Objekts

über die entsprechende Komponente einer Benutzerschnittstelle. Ein Objekt, das den Zugriff auf sein *Control*-Objekte unterstützt, implementiert die *Controls*-Schnittstelle. Die *Controls*-Schnittstelle besitzt Methoden, die das entprechende *Control*-Objekt zurückgeben.

CachingControl ermöglicht Überwachung und Anzeige des Downloadfortschritts.
GainControl ermöglicht die Regulierung der Lautstärke, wie das Niveau und das Stummschalten.
StreamWriterControl ermöglicht die Limitierung der Grösse des erzeugten Stromes.
FramePositioningControl und *FrameGrabbingControl* erweitern Player und Prozessoren um framespezifische Fähigkeiten. *FramePositioningControl* für die genaue Positionierung innerhalb des Player- oder Processor-Objekts. *FrameGrabbingControl* für die Aufnahme eines Standbildes aus dem Videostrom.
FormatControl liefert für ein Objekt, das ein bestimmtes Format hat, Methoden, die dieses Format abfragen und ändern können.
TrackControl ermöglicht die Kontrolle darüber, welche Prozesse ein *Processor*-Objekt auf eine bestimmte Spur von Mediendaten ausführt. *TrackControl* ermöglicht auch die Bestimmung, welche Formatumwandlung auf welcher Spur ausgeführt werden soll.
PortControl und *MonitorControl* ermöglichen die Kontrolle über den Erfassungsprozess. *PortControl* definiert Methoden für die Steuerung der Ausgabe des Erfassungsgeräts. *MonitorControl* ermöglicht eine Vorschau der Mediendaten während der Erfassung oder Codierung.
BufferControl ermöglicht Steuerung der Pufferung, die von JMF-Objekten ausgeführt wurde, auf Benutzerebene.
Zur Steuerung der Hardware- oder Software-Codierer und Decodierer, definiert das JMF mehrere Codec-Controls.

3 Die Präsentation von Mediendaten

Die *Controller*-Schnittstelle modelliert den Präsentationsprozess. Der *Controller* definiert den Steuerungsmechanismus für ein Objekt, das zeitabhängige Medien steuert, präsentiert und erfasst. Er definiert die Phasen, die ein Medienkontrolleur durchlaufen muss und liefert einen Mechanismus für die Steuerung der Übergänge zwischen den Phasen. Für mehr Informationen über die "Phasen der Verarbeitung" siehe Abschnitt 4.1.
Der *Controller* sendet verschiedene MediaEvents, um Nachrichten über Änderungen in seinem Zustand zu liefern. Eine Klasse, deren Objekte ein Ereignis vom *Controller* empfangen sollen, muss die *ControllerListener*-Schnittstelle implementieren. Für mehr Informationen über Ereignisse des Controllers siehe Abschnitt 3.4

Die JMF API definiert zwei Typen des Controllers: Player und Processor.

3.1 Player

Eine Datenquelle liefert einen Eingabestrom von Mediendaten an den Player. Der Player verarbeitet den Eingabestrom von Mediendaten und gibt diesen zu einer genauen Zeit wieder. Dabei hängt das Wiedergabegerät von dem Typ der Mediendaten, die präsentiert werden sollen, ab.
Der Player befindet sich in einem von sechs Zuständen. Die *Clock* definiert zwei primäre Zustände: *Stopped* und *Started*. Um die Verwaltung der Betriebsmittel zu erleichtern, spaltet der *Controller* den Zustand *Stopped* in fünf Zusände auf : *Unrealized*, *Realizing*, *Realized*, *Prefetching* und *Prefetched*. Im normalen Ablauf geht der *Player* durch alle Zustände bis er den Zustand *Started* erreicht hat:

- Wenn ein *Player* erzeugt wird, ist er im Zustand *Unrealized*.

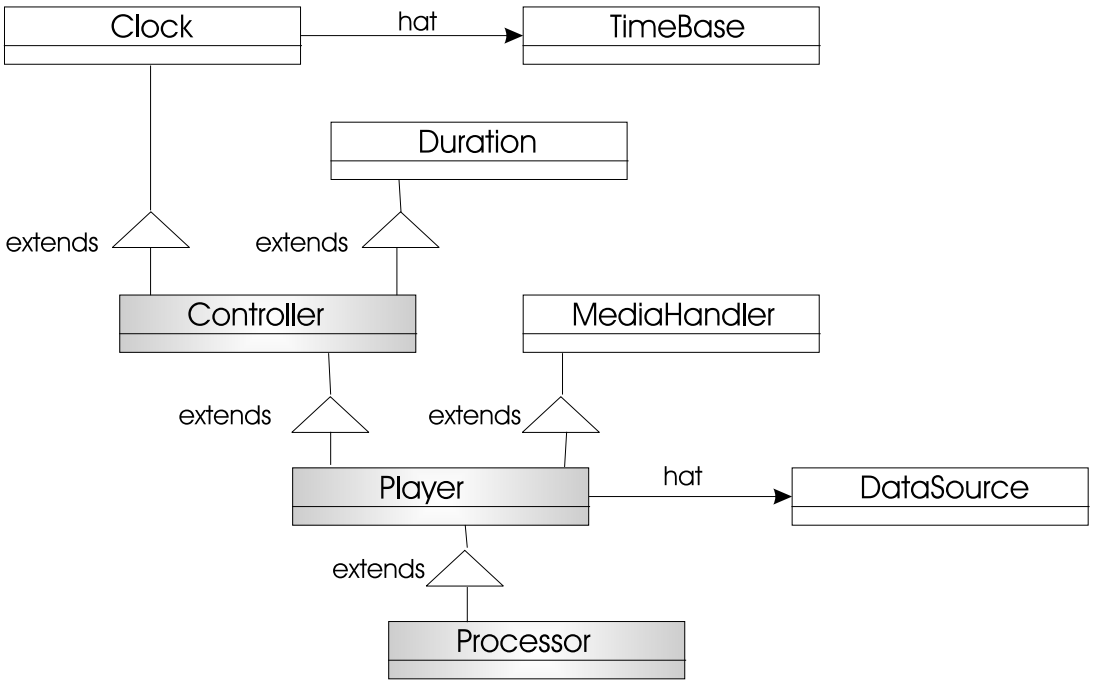

Abbildung 2: JMF Controllers

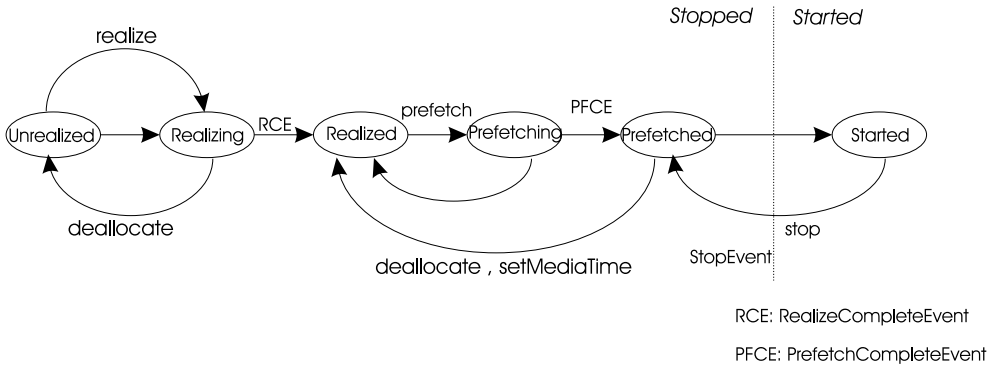

Abbildung 3: Zustände des Players

- Wenn *realize* aufgerufen wird, wechselt der *Player* von dem Zustand *Unrealized* in den Zustand *Realizing*. Hier bestimmt der *Player* seine Betriebsmittel und seine Anforderungen.

- Wenn der *Player* mit dem *Realizing* fertig ist, wechselt er in den Zustand *Realized*. Der *Player* ist jetzt informiert über seine Betriebsmittel und über den Medien-Typ, den er präsentiert.

- *Player* wechselt in den Zustand *Prefetching* durch den Aufruf von *prefetch*. Hier lädt der *Player* seine Mediendaten vor, und bekommt Betriebsmittel für die exklusive Nutzung. In diesem Zustand unternimmt der *Player* alles, was er für das Abspielen braucht.

- Wenn der *Player* mit dem *Prefetching* fertig ist, wechelt er in den Zustand *Prefetched*. Der *Player* ist jetzt bereit für den Start.

- Der Aufruf von *start* bringt den *Player* in den Zustand *Started*. Ein *Started-Player* die time-base Zeit ist jetzt abgebildet und die Uhr des *Player* läuft, auch wenn der *Player* auf einen Event wartet, um die Präsentation zu beginnen.

Der *Player* sendet *TransitionEvents*, wenn er in den Zustand wechselt. Der *ControllerListener* ermöglicht, den Zustand des *Player* abzufragen, und entprechend darauf zu reagieren.

3.2 Prozessoren

Ein *Player* unterstützt keine Steuerung der Verarbeitung, die er durchführt oder wie er die Mediendaten wiedergibt. Ein *Processor* ist ein spezieller Typ von *Playern*, der die Steuerung der Verarbeitug der Medienströme unterstützt.
Ausser die Wiedergabe von Mediendaten an die Präsentationsgeräte, kann ein *Processor* Mediendaten durch eine Datenquelle so ausgeben, dass sie von einem anderen *Player* oder *Processor* präsentiert werden können, oder dass sie von einem anderen *Processor* zusätzlich verarbeitet werden oder an andere Ziele gereicht werden. Für weiter Infromationen über die Verarbeitung von Mediendaten siehe Abschnitt 4.

3.3 Die Steuerung der Präsentation

Ein *Player* oder ein *Processor* kann zusätzliche zu der standard Präsentationssteuerung die Einstellung des Playbackvolumen unterstützen. In diesem Fall, kann man seine *GainControl* durch den Aufruf von *getGainControl* abfragen. Das *GainControl*-Objekt sendet ein *GainChangeEvent*, wenn die Verstärkung verändert wird. Eine Klasse, deren Objekte auf veränderungen in der Versärkung reagieren sollen, muss die *GainChangeListener*-Schnittstelle implementieren.
Zusätzliche angepasste Steuerungstypen können von speziellen *Players* oder *Processors* unterstützt werden, um andere Steuerungsverhalten zu ermöglichen. Mit der Methode *getControls* kann man auf diese Steuerungen zugreifen.

3.4 Ereignisse des Controllers

Die *ControllerEvents*, die ein *Controller* wie *Player* oder *Processor* sendet, können wie folgt kategorisiert werden:

- *Änderung* - Ereignisse dieser Kategorie - wie *RateChangeEvent*, *DurationUpdateEvent*, oder *FormatChangeEvent* - weisen darauf hin, dass die Attribute des *Controller*s sich verändert haben, meistens durch einen Methodenaufruf.
- *Übergang* - *TransitionEvents* ermöglichen die Reaktion auf Veränderungen im Zustand des *Controller*s. Ein *Player* sendet *TransitionEvent*, wenn er von einem Zustand in den Anderen ewechselt hat.
- *Geschlossen* - *ControllerClosedEvents* werden gesedet, wenn ein *Controller* heruntergefahren wird. Ab diesen Moment ist der *Controller* unbrauchbar. Das *ControllerErrorEvent* ist ein spezieller Fall von *ControllerClosedEvent*.

4 Die Verarbeitung von Mediendaten

Ein *Processor* ist ein *Player*, der eine Datenquelle als Eingabe annimmt, die Mediendaten der Datenquelle verarbeitet, und die verarbeitet Mediendaten an ein Präsentationsgerät oder in eine Datenquelle ausgibt. Die Datenquelle kann wiederum als Eingabe für einen anderen *Player* oder als Eingabe für eine Datensenke.

Ein *Processor* lässt dem Anwedungsentwickler zu, die Arte der Verarbeitug der Mediendaten zu definieren, was die Anwendung von Effekten, Mischen, und Komponieren in Echtzeit möglich macht.

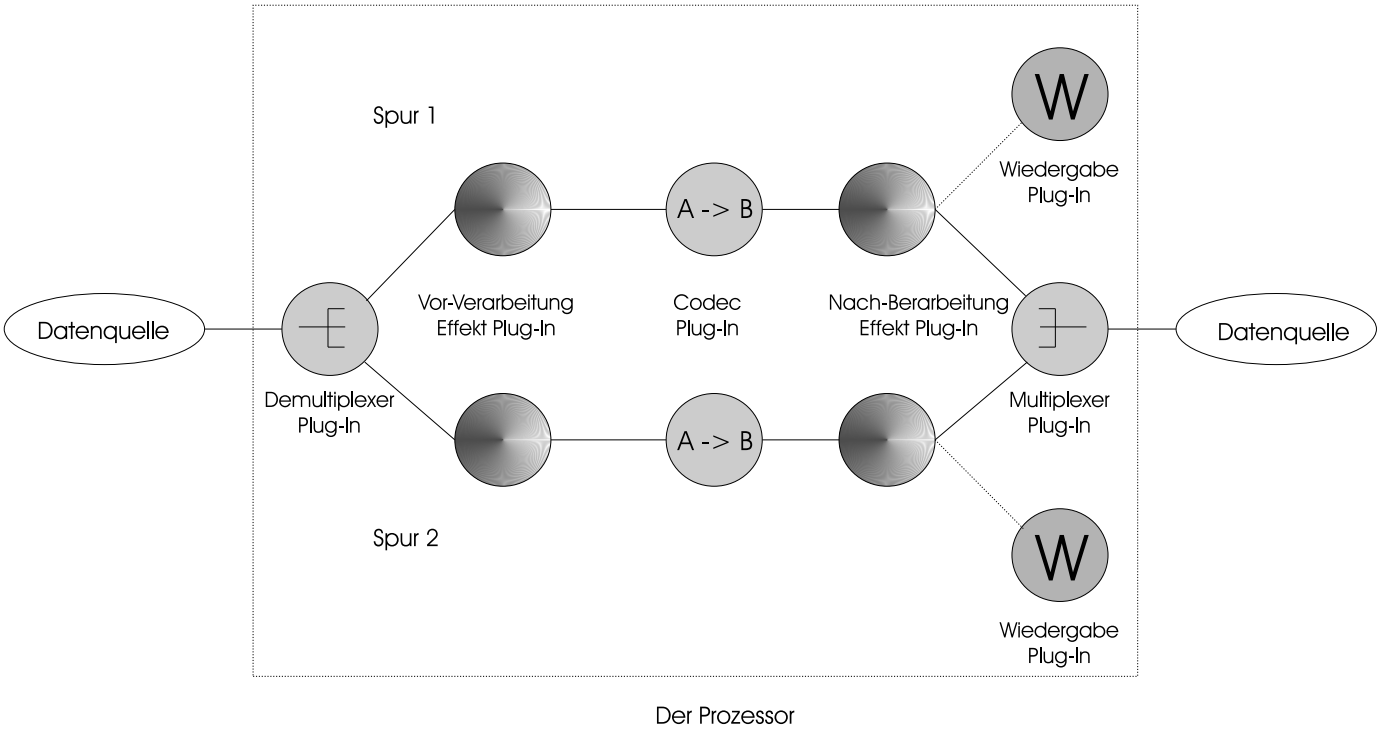

Abbildung 4: Phasen des Processors

4.1 Phasen der Verarbeitung

Ein *Processor* hat die folgenden sechs Phasen der Verarbeitung:

- *Das Demultiplexen* - Der Eingabestrom wird analysiert. Falls der Strom mehrere Spuren hat, werden die Spuren extrahiert und getrennt ausgegeben.
- *Die Vor-Verarbeitung* - Hier werden Effektalgorithmen auf die extrahierten Spuren angewendet.
- *Die Umkodierung* - Die Spuren von Mediendaten werden in dieser Phase von einer Format in die Andere konvertiert, wie z.B. Kodierung und Dekodierung.
- *Die Nach-Bearbeitung* - Effektalgorithmen werden hier auf die dekodierte Spuren angewendet.
- *Das Multiplexen* - Umkodierte Spuren werden jetzt in einen einzelnen Ausgabestrom kompiniert, wie z.B. getrennte Audio-/ und Video-Spuren können in einen einzelnen MPEG-1-Datenstrom kompiniert werden. Mit *setOutputContentDescriptor* wird der Datentyp der Ausgabestrom bestimmt.
- *Die Wiedergabe* - Die Medien werden hier dem Benutzer präsentiert.

In jeder Phase übernimmt eine andere Komponente - ein JMF Plug-In - die Verarbeitung. Wenn der *Processor TrackControls* untersützt, dann kann der Entwickler das Plug-In für die Bearbeitung einer bestimmten Spur wählen. Das JMF hat die folgenden fünf Typen von Plug-Ins:

- *Demultiplexer* - Dieser Typ analysiert die Medienströme. Wenn der Strom multiplext ist, werden die einzelnen Spuren extrahiert.
- *Effect* - Dieser Typ führt Spezialeffektbearbeitung auf eine Spur aus.
- *Codec* - Dieser Typ von Plug-Ins kodiert und dekodiert die Spuren.
- *Multiplexer* - Plug-Ins diesen Typs Kombinieren mehrere Spuren in einen Strom und geben diesen als eine Datenquelle aus.
- *Renderer* - Dieser Typ bearbeitet die Mediendaten in einer Spur und liefert sie an Ziele wie Monitore oder Lautsprecher.

4.2 Zustände des Prozessors

Ein *Processor* hat zwei zusätzliche Reservezustände, *Configuring* and *Configured*, die der *Processor* passiert, bevor er in den Zustand *Realizing* wechselt.

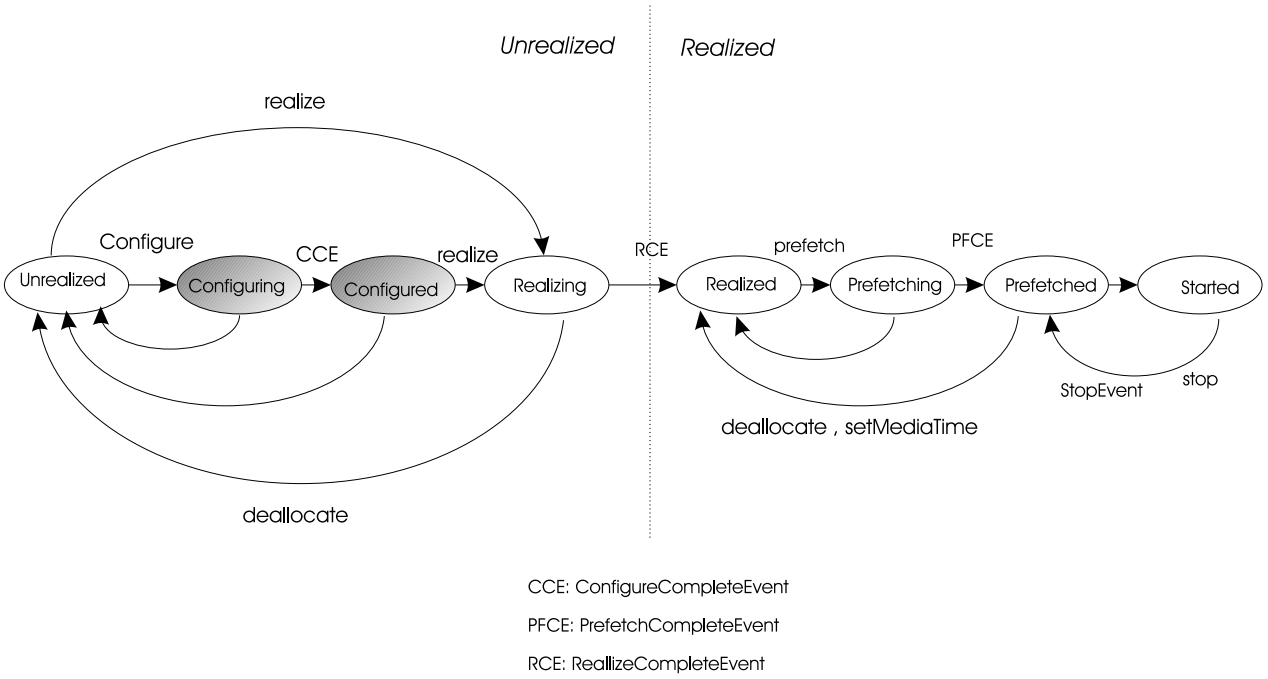

Abbildung 5: Zustände des Prozessors

Der *Processor* betritt den Zustand *Configuring*, wenn *configure* aufgerufen wird. In diesem Zustand verbindet sich der *Processor* mit der Datenquelle, demultiplext den Eingabestrom und greift auf Informationen über das Format der Eingabedaten zu.
Der *Processor* wechselt in den Zustand *Configured*, wenn er mit der Datenquelle verbunden ist und das Datenformat festgelegt ist. Wenn der *Processor* diesen Zustand erreicht hat, wird *ConfigureCompleteEvent* gesendet. Während der *Processor* in dem Zustand *Configured* ist, kann man mit *getTrackControls* ein *TrackControl*-Objekt für eine bestimmte Spur erhalten. Das *TrackControl*-Objekt ermöglicht die Beschreibung der Bearbeitung der Mediendaten, die der *Processor* ausführt.
Wenn *realize* aufgerufen wird, wechselt der *Processor* dann in den Zustand *Realized*. Wenn aber realize direkt für einen *Unrealized Processor* aufgerufen wird, wechselt der *Processor* in *Configuring* und in *Configured* und dann in den Zustand *Realized*. In diesem Fall kann man keine Bearbeitugsoptionen durch *TrackControls* konfigurieren. Ein *Realized Processor* ist vollständig hergestellt.

4.3 Steuerung der Verarbeitung

Was der *Processor* an Verarbeitungstätigkeiten auf eine Spur ausführt, kann man durch *TrackControl* steuern. Durch *TrackControl* kann man explizit die Effekt- , Codec-, und Wiedergabe-Plug-Ins, die auf die Spur angewendet werden, wählen. Informationen über installierte Plug-Ins erhält man durch Abfragen des *PlugInMangagers*.
Der Aufruf der Methode *getControls* des *TrackControls* liefert die Steuerung der entprechenden Spur. Diese Methode gibt die *Codec*-Steuerungen, die für diese Spur vorhanden sind, wie *BitRateControl* und *QualityControl*.
Ist das Format der Ausgabe bekannt, so kann das Format mit *setFormat* spezifiziert werden, und die Wahl eines geeigneten *Codec* und Wiedergeber wird dann dem *Processor* überlassen.

4.4 Ausgabe der verarbeiteten Mediendaten

Die Ausgabe des *Processor*s erhält man in Form einer Datenquelle durch den Aufruf der Methode *getDataOutput*. Diese Datanquelle kann als Eingabe für einen aderen *Player* oder *Processor* benutzt werden oder als Eingabe für eine Datensenke. Für mehr Informationen über Datensenken, siehe Abschnitt 6
Die vom *Processor*-Objekt ausgegeben Datanquelle kann vom Typ *PushDataSource*, *PushBufferDataSource*, *PullDataSource*, oder PullBufferDataSource sein.
Ein *Processor*-Objekt kann, statt eine Datenquelle auszugeben, die verarbeiteten Daten wiedergeben. In diesem Fall ist der *Processor* hauptsächlich ein konfigurierbarer *Player*

5 Erfassung von Mediendaten

Die Erfassugsgeräte für Multimedia können als eine Quelle für Multimediadaten dienen. Ein Mikrofon, z.B., kann originale Audio-Eingaben erfassen, oder ein Erfassungsboard für digitales Video, kann digitales Video von einer Kamera liefern. Solche Erfassungsgeräte werden vereinfacht, als Datenquellen gesehen. So wird z.B. ein Gerät, das rechzeitige Lieferung von Daten unterstützt, als *PushDataSource* gehandelt. Jeder Typ von Datenquellen kann als Erfassungsdatenquellen benutzt werden, wie *PushDataSource*, *PushBufferDataSource*, *PullDataSource*, oder *PullBufferDataSource*.

6 Speicherung und Übertragung von Mediendaten

Eine DatenSenke (*DataSink*) wird benutzt, um Mediendaten von einer Datenquelle zu lesen, und diese an ein Zielgerät - üblicherweise kein Präsentationsgerät - wiederzugeben. Bestimmte Datensenken können Daten in eine Datei schreiben, Daten über das Netzwerk schreiben, oder als ein *RTP-Broadcaster* funktionieren.Für mehr Informationen über die Benutzung von Datensenken als RTP-Broadcaster, siehe „Transmitting RTP Data With a Data Sink", in [Sun99].
Ein DataSink kann StreamWriterControl benutzen, um zusätzliche Steuerung des Schreibens in Dateien zu unterstützen.

6.1 Steuerung der Speicherung

Eine *DataSink* sendet ein *DataSinkEvent*, um über seinen Zustand zu berichten. *DataSinkEvent* wird mit einem Erklärungscode - Grund des Ereignises - oder der *DataSink* sendet einen der Subtypen des *DataSinkEvents*:

- *DataSinkErrorEvent*, wenn Fehler auftreten, während der *DataSink* Daten schreibt.
- *EndOfStreamEvent*, um zu berichten, dass der Strom vollständig und erfolgreich geschrieben worden ist.

Um auf die Ereignisse, die *DataSink* gesendet hat, zu reagieren, muss die *DataSinkListener*-Schnittstelle implementiert werden.

7 Erweiterbarkeit des JMF

JMF kann durch die Implementierung von agepassten Plug-Ins, Behandlungsprozeduren, und Datenquellen erweitert werden.

7.1 Implementierung von Plug-Ins

Die Implementierung einer der JMF Plug-In-Schnittstellen ermöglicht die Manipulation und den direkten Zugriff auf die Mediendaten, die vom *Processor* bearbeitet werden.
Angepasste *Codec*-, *Effect*-, und *Renderer*-Plug-Ins sind für den *Processor* durch die *TrackControl*-Schnittstelle erreichbar. Um Plug-Ins für *Processors* verfügbar zu machen, müssen diese mit dem *PlugInManager* registriert werden. Durch den Aufruf von *getPlugInList* liefert der *PlugInManager* eine Liste der registrierten Plug-Ins.

7.2 Implementierung von Behandlungsprozeduren und Datenquellen

Falls die JMF Plug-In API die nötige Flexibilität nicht liefert, kann man die JMF Kern-Schnittstellen direkt implementieren, wie *Controller*, *Player*, *Processor*, *DataSource*, und *DataSink*. Man kann z.B. einen für ein bestimmtes Foramt optimierten Höchstleistung-*Player* implementieren.
Der Mechanismus des *Managers* für das Anlegen von *Player*-, *Processor*-, *DataSource*-, und *DataSink*-Objekten ermöglicht benutzung von agepassten Implementierungen dieser Schnittstelle.

Literatur

[Stei00] Ralf Steinmetz. *Multimedia-Technologie.* Springer. 2000.

[Sun99] Sun. *Java Media Framework API Guide*, 1999.

Abbildungsverzeichnis

IP-Telefonie - Standards, Anwendungen und Analyse

Andreas Mitschele

Kurzfassung

Seit ihrer Markteinführung im Jahre 1995 durch die israelische Firma Vocaltec hat die IP-Telefonie mit rasanten Wachstumsraten auf sich aufmerksam gemacht. Alle großen Firmen im Telekommunikationsgeschäft beschäftigen sich mit der relativ neuen Technologie und bemühen sich um eine gute Positionierung im Wettbewerb. Diese Arbeit soll einen Überblick über den breiten Themenbereich geben. Dabei werden zunächst technische Grundlagen der verbreiteten Standards sowie ihre Interoperabilität dargestellt. Weitere Themen sind mögliche Anwendungen der IP-basierten Telefonie und eine Analyse ihrer Schwächen gegenüber traditionellen Telefonnetzwerken. Im Anschluss wird die IP-Telefonie in das Marktumfeld mitsamt regulatorischer und rechtlicher Problemstellungen eingeordnet. Zuletzt erfolgt ein abschliessender Ausblick mit Zukunftsprognosen und Feldern, in denen noch Handlungsbedarf besteht.

1 Einleitung

1.1 Begriffsklärung

Mit IP-Telefonie wird die Übertragung von Sprach-, Fax- und verwandten Diensten über paketorientierte Datennetze, die auf dem Internet Protocol (IP) basieren, bezeichnet. Sie lässt sich weiter unterteilen in *Internet-Telefonie* über das weltumspannende Internet und *Voice-over-IP* (kurz: VoIP) über geschlossene Netze (z.B. Intranets). Die anfängliche Beschränkung auf Sprachdaten wurde auf Multimedia-Daten wie beispielsweise Video für Video-Conferencing ausgeweitet.

Einzig entscheidend dabei ist, dass die IP-Protokollfamilie zugrundeliegt. IP-Telefonie ist also durchaus unabhängig vom öffentlichen Internet funktionsfähig, beispielsweise innerhalb eines Firmennetzes [ITU00].

1.2 Ausprägungsformen

Grundsätzlich kann IP-Telefonie weiter eingeteilt werden in Kommunikation von

- PC zu PC
- PC zu Telefon und umgekehrt
- Telefon zu Telefon

Von PC zu PC erfolgt die Kommunikation komplett über das zugrunde liegende Datennetz, zumeist das öffentliche Internet. Anrufe müssen üblicherweise vereinbart werden und sind

nur möglich, wenn beide Gesprächspartner online sind. Die Adressierung erfolgt über die IP-Adressen der Rechner.

Wird ein herkömmliches Telefon oder Faxgerät vom PC aus angewählt, gehen die Daten zunächst per IP über das Datennetz und müssen dann an einem Gateway für das PSTN (Public-Switched-Telephone-Network) übersetzt werden. Aus der Sicht des Providers gestaltet sich diese Art der Telefonie komplizierter, denn es müssen Abrechnungs- und Routingvereinbarungen mit konventionellen PSTN-Anbietern getroffen werden.

Technisch ähnlich funktioniert die Kommunikation von Telefon zu Telefon, nur müssen hier die Daten, nachdem sie über ein Gateway ins IP-Netz eingespeist worden sind, wieder auf das PSTN übertragen werden. Dabei haben die meisten VoIP-Dienste nur noch wenig mit dem Internet zu tun, denn sie laufen parallel zum PSTN und dessen Gebührensystem. Von seiten der nötigen Kooperationsvereinbarungen und Anzahl der weltweiten Präsenzpunkte gestaltet sich dieser Ansatz am komplexesten. Dennoch wird dieser als einer der erfolgsversprechendsten angesehen, weil Benutzer es vorziehen, weiterhin mit dem normalen Telefon zu telefonieren.

Der Anruf eines PCs vom Telefon kommt in der Praxis eher selten vor, denn nur wenige Rechner im Internet verfügen über eine feste IP-Adresse, die zur Adressierung nötig ist [ITU00]. Auf Basis dieser Varianten kann VoIP nun in einer Vielzahl von praktischen Anwendungen und Zusatzservices eingesetzt werden, die in Abschnitt 3 vorgestellt werden.

1.3 Historie der IP-Telefonie

Erste Ansätze zur Übertragung von Sprachdaten über das Internet gab es bereits Anfang der neunziger Jahre, allerdings brachte erst Vocaltec 1995 eine brauchbare Lösung auf den Markt, die zur freien Verfügung stand. Es war anfangs nur möglich im Halbduplex-Betrieb - wie beim CB-Funk - abwechselnd zu sprechen. Im Jahr 1997 begannen erste kommerzielle IP-Telefonieanwendungen im Bereich Firmennetze Fuss zu fassen [Stüt01]. Trotz massiven Widerstands von seiten traditioneller Telefongesellschaften in den USA, wurde Internettelefonie von der US-Behörde für Fernmeldewesen zugelassen. Damit war in einem der wichtigsten Märkte der Weg für viele Anbieter von VoIP-Lösungen geebnet [ITU00].

2 Technische Grundlagen

Bevor im folgenden auf spezielle Eigenschaften der IP-Telefonie näher eingegangen wird, erfolgt eine Einordnung in traditionelle Telefonnetzwerke. Anschliessend wird die Internet-Protocol(IP)-Familie noch kurz vorgestellt.

2.1 Heutige Telefonnetze

Alexander Graham Bell legte im Jahre 1876 den Grundstein für die heutigen PSTNs, als es ihm zum ersten Mal gelang, Sprache über einen Schaltkreis zu übertragen. Da alles, was wir hören, in analoger Form vorliegt, wurde das Telefonnetz bis vor wenigen Jahrzehnten auf Basis analoger Technik betrieben. Wegen hoher Anfälligkeit gegen Leitungsrauschen bei der Verstärkung der Signale ging man auf digitale Signalisierung über. Dabei wird mittels PCM (Pulse-Code-Modulation) das analoge Sprachsignal abgetastet und in eine digitale Folge von Nullen und Einsen zerlegt. Dieser kontinuierliche Strom wird über das Telefonnetz geschickt und in den Repeatern nicht nur verstärkt, sondern auch regeneriert.

Problematisch bei der “alten“ Telefontechnik ist, dass sie als Netz für Sprachübertragung entwickelt wurde. Inzwischen hat aber der Datenverkehr den Sprachverkehr auf den Leitungen

überholt. Während bis heute die Datenübertragung noch eher über das Telefonkabel erfolgt, so geht man davon aus, dass zukünftig Telefonieren über Datenleitungen stattfinden wird, mittels IP-Telefonie. Damit kommt man auch dem Ziel der Konvergenz von Daten, Voice und Video einen großen Schritt näher. Als eine der größten Herausforderungen stellt sich hierbei die Tatsache, dass IP-Netzwerke auf der Übermittlung von Datagrammen beruhen und daher von Natur aus nicht für Realtime-Übertragung geeignet scheinen [DaPe00].

2.2 Überblick über IP

Mit dem Begriff IP nimmt man meist Bezug auf eine komplette Protokollfamilie, die in sogenannten IP-Netzen Daten als Pakete überträgt. IP kann in vielen verschiedenen Netzen zum Einsatz kommen und wurde so weltweit praktisch allgegenwärtig. Dies stellt eine sehr solide Basis für die Ausbreitung von VoIP dar.

Auf IP setzen die beiden Transportprotokolle TCP (Transmission Control Protokoll) und UDP (User Datagram Protokoll) auf. Die verbindungsorientierte, gesicherte Ende-zu-Ende-Verbindung von TCP wird beim VoIP meist für Signalisierungsprotokolle, d.h. vor allem für die Anrufeinrichtung, eingesetzt. Aufgrund des großen Overheads von TCP und aufgrund der Tatsache, dass bei einer Sprachübertragung eine kurze Latenzzeit (siehe Abschnitt 4.1.1) wichtiger ist als gesicherte Pakete, kommt zur Gesprächsübermittlung das verbindungslose und ungesicherte UDP zum Einsatz. Es ist einfacher gebaut und muss für eine korrekte Verarbeitung der Pakete, z.B. der Reihenfolge, durch andere Routinen ergänzt werden [DaPe00].

Den speziellen Anforderungen von verzögerungsempfindlichem Verkehr bei der Übertragung kam die IETF im RFC 1889 vom Januar 1996 durch RTP (Realtime Transport Protokoll) nach. Dieses operiert in der Anwendungsebene und ergänzt die TCP/IP-Datenpakete durch eine Sequenznummer zur Überprüfung der Reihenfolge sowie einen Zeitstempel, um die Verzögerung bei der Ankunft der Pakete zu bestimmen. Die verbreiteten Standards für IP-Telefonie greifen auf RTP zurück. [MiMi98].

2.3 Standards

In den letzten Jahren wurden eine Reihe von Standards für die IP-Telefonie entwickelt. Vor allem H.323 und SIP haben sich dabei etablieren können.

2.3.1 H.323

Der Standard H.323 wurde im Oktober 1996 von der ITU-T (International Telecommunication Union - Telecommunication Standardization Sector) für die Übertragung von Audio, Video und Daten über IP-Netzwerke spezifiziert. Dieser Standard ist am weitesten verbreitet und wird beispielsweise bei PC-Konferenzen über Microsoft-Netmeeting eingesetzt. Er dient als Kommunikationsschnittstelle für Produkte verschiedener Hersteller.

In einem H.323-System gibt es vier Netzbestandteile (Abbildung 1), die für die Kommunikation notwendig sind. Die Endgeräte werden als *Terminals* bezeichnet, die vor allem Sprachdienste aber auch Video- und Datenkommunikation unterstützen. Sie setzen sich aus einer System-Kontrolleinheit, einer Medien-Übertragungs-, einer Audio-Codec- und einer paketbasierten Netzwerkschnittstelle zusammen. Die Übergänge zwischen unterschiedlichen Netztypen, z.B. zwischen LAN und ISDN, werden durch *Gateways* realisiert. Obwohl es herstellerspezifische Unterschiede gibt, konvertieren sie grundsätzlich Übertragungsformate und Kommunikationsprozeduren zwischen verschiedenen Netzwerkstandards (z.B. ISDN, GSM). Das Bandbreiten-Management wird von den sogenannten *Gatekeepern* übernommen. Außerdem

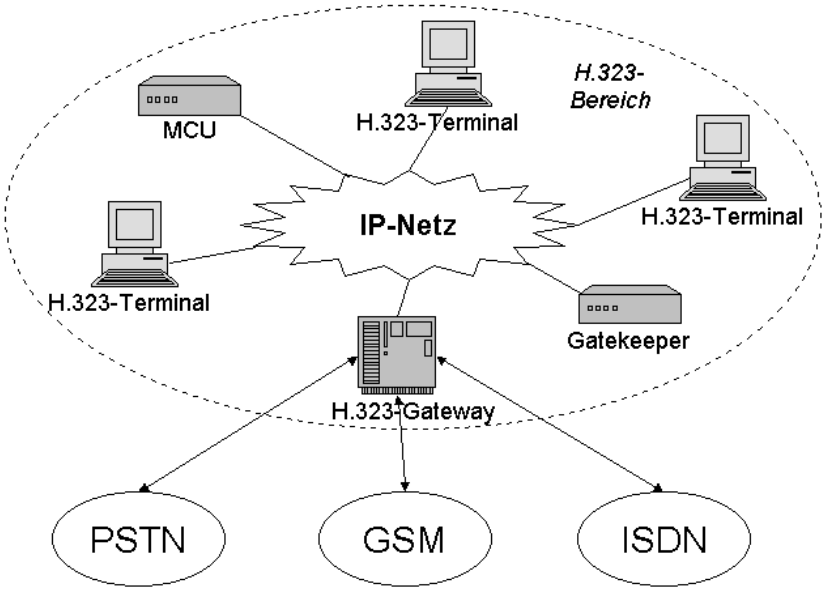

Abbildung 1: H.323-Architektur

übersetzen sie die Adressen und führen Zugangskontrollen durch. Die *Multipoint-Controller-Unit (MCU)* schließlich besteht aus einem Multipoint-Controller, der Konferenzen mit mehr als zwei Endpunkten unterstützt, und einem Multipoint-Prozessor, der empfangene Audio-, Video- und Datenströme an die Teilnehmer der Konferenz weiterschickt.

Der Name H.323 wird stellvertretend für eine komplette Protokollfamilie (Abbildung 2) benutzt. Dabei wird zur Signalisierung der Standard H.225 verwendet, der ab der Version 2 auch UDP als Transportprotokoll unterstützt. Dieser wiederum besteht aus den Subprotokollen RAS (Registration, Admission & Status), das vom Gatekeeper zur Signalisierung und von den Terminals zur Anmeldung benutzt wird, und Q.931 für die Anrufkontrolle. Über H.245 erfolgt die Verbindungskontrolle und durch H.235 wird Datensicherheit und Verschlüsselung ergänzt. Die Gruppe G.7x ist verantwortlich für die Codierung und Komprimierung der Sprache, H.26x für Video und T.x für Datenanwendungen. Über H.450 werden zusätzliche Dienste definiert, die im Moment noch das große Problem der VoIP darstellen, da viele Leistungsmerkmale z.B. der ISDN-Telefonie noch nicht eingebaut sind. Bei der Anrufeinrichtung werden sowohl gesicherte (H.245 Kontrolle, Signalisierung mit TCP) als auch ungesicherte Kanäle (Daten, Sprache, Video, UDP) benutzt.

Probleme haben sich bei H.323 durch lange Anrufeinrichtungszeiten, einen großen Overhead und Skalierbarkeitsbeschränkungen ergeben.

2.3.2 SIP

Die IETF (Internet Engineering Task Force) hat mit dem Request for Comments (RFC) 2543 das SIP (Session Initiation Protocol) entworfen. Es arbeitet als textbasiertes Signalisierungs-Protokoll auf der Applikationsschicht und dient zur Einrichtung, Unterhaltung und Beendigung von Multimedia-Sitzungen. Da es im Vergleich zu H.323 relativ einfach aufgebaut ist, hat es die Vorteile eines schnelleren Anrufaufbaus und weist zudem eine höhere Flexibilität auf [FKSS01]. Auch wegen relativ einfach zu realisierender Erweiterungsmöglichkeiten wird es

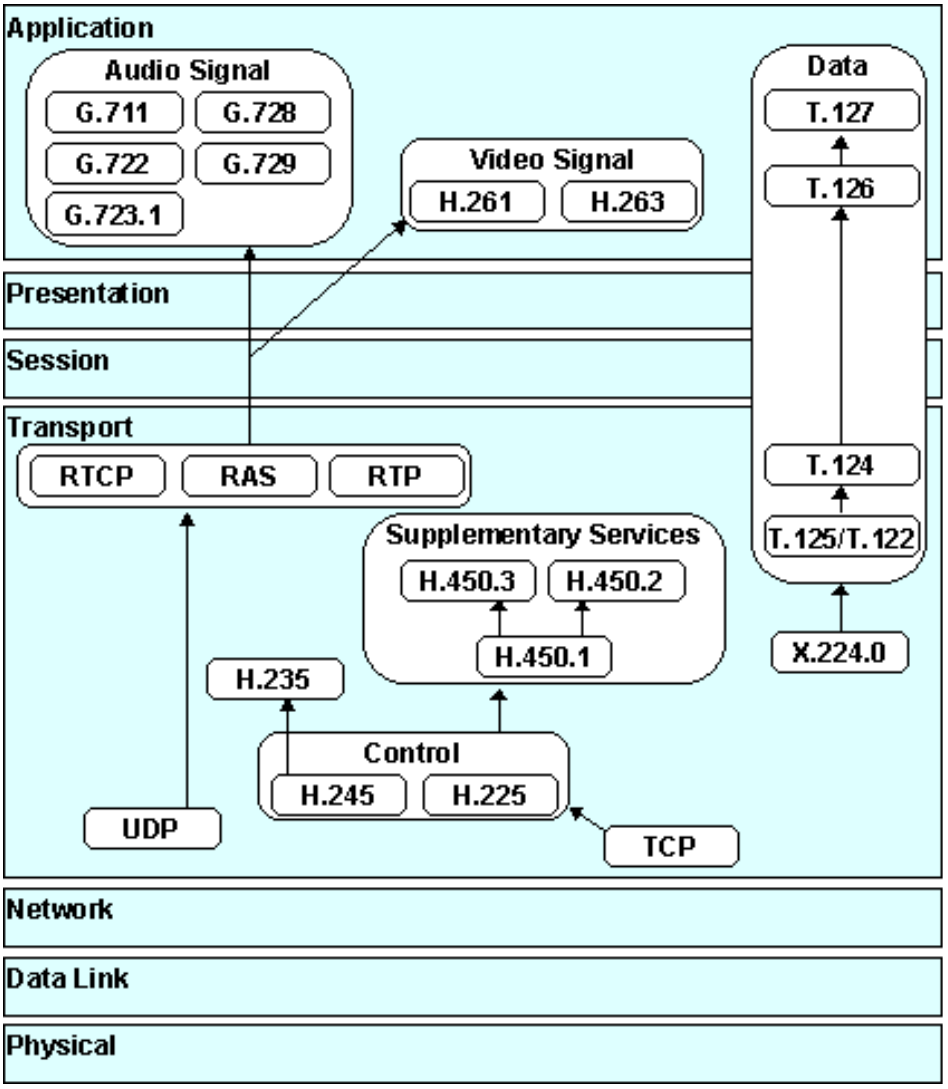

Abbildung 2: Protokollfamilie H.323

als ernsthafte Konkurrenz für H.323 gesehen und in der Praxis als Basis für Carrier Networks und IP-Telefone eingesetzt.

Im wesentlichen besteht ein SIP-Netzwerk (Abbildung 3) aus zwei Komponenten, Benutzeragenten und Netzwerk-Servern. Benutzeragenten sind Client-Applikationen auf Endsystemen und lassen sich ihrerseits unterteilen in *User Agent Client (UAC)* und *User Agent Server (UAS)*. Der UAC schickt SIP-Anfragen und handelt als anrufender Agent des Benutzers, während der UAS als angerufener Agent des Benutzers Anfragen verarbeitet und beantwortet. Bei den Netzwerk-Servern unterscheidet man zwei mögliche Modi, nämlich Proxy- und Redirect-Server.

SIP ist ein Client-Server-Protokoll und daher gibt es nur zwei Arten von Meldungen. Anfragen (Request) kommen vom Client an den Server und beinhalten z.B. INVITE, ACK oder BYE. Antworten (Response) gehen dann vom Server an den Client und können informieren, bestätigen oder Fehler anzeigen. Der typische Ablauf einer SIP-Konferenz gestaltet sich wie folgt: Die Konferenz wird entweder per Session-Announcement-Protocol (SAP) bekannt gegeben oder Teilnehmer werden per SIP von anderen eingeladen. Mittels SDP (Session Description Protokoll) werden die Eigenschaften der Sitzung beschrieben (Zeit, Multimediafähigkeit, etc.). Die Anrufkontrolle kann durch den Proxy-Server durchgeführt werden, der die Anfragen

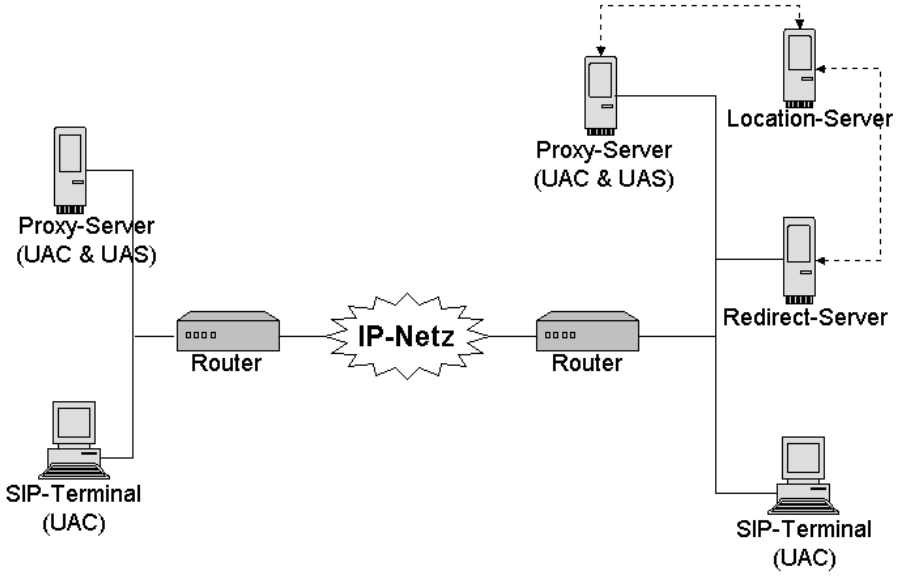

Abbildung 3: SIP-Architektur

von den Endsystemen erhält und mit Hilfe der Adressen im Location Server verarbeitet. Im Redirect Modus sendet der SIP-Server die Anfrage nicht an den Gerufenen weiter, sondern schickt dessen Adresse an den Anrufenden zurück [DaPe00].

2.4 Interoperabilität

Auch wenn IP-Netze langfristig traditionelle Sprachnetze ablösen, wird es noch viele Jahre lang Telefonate geben, die von PSTNs ausgehen. Daher stellt die problemlose Zusammenarbeit der IP-Telefonie mit PSTNs ebenso wie die Zusammenarbeit der Standards untereinander einen entscheidenden Erfolgsfaktor dar.

Der heute gebräuchliche Einsatz von Gateways, die Anrufe zwischen den verschiedenen Netzen einfach übersetzen, macht zwar Sprachverkehr problemlos möglich, scheitert aber meist an der Unterstützung zusätzlicher Dienste von INs (Intelligent Networks), wie der Übermittlung der Rufnummer des Anrufers. Aus diesem Grund versucht man durch aktuelle Forschungsarbeiten die Telefonie "medienneutral" zu gestalten, ähnlich wie beim Internet-Ansatz [ITU00].

Die beiden meistverbreiteten Standards H.323 und SIP stellen vollkommen verschiedene Lösungen zum selben Problem dar. Während H.323 eher vom traditionell leitungsvermittelten Netz ausgeht, basiert SIP auf einer weniger komplizierten Lösung, ähnlich HTTP. SIP eignet sich eher als Internet-Zusatzdienst und bietet sich für Internet-Provider an, während H.323 auch alleine arbeiten kann. Obwohl beide Standards in hohem Maße interoperabel sind, ergeben sich in der Praxis Probleme [SiSc00]. Im Februar 2001 hat die IETF mit dem Internet-Draft "SIP-H.323-Interworking" eine Funktion vorgestellt, die eine Ende-zu-Ende-Kommunikation zwischen beiden Protokollen zur Verfügung stellt. Obwohl beide Protokolle von der Funktionalität her ähnlich sind und sich aufeinander zubewegen, müssen dennoch Anpassungen durchgeführt werden. Es müssen die Signalisierungsprotokolle und die Sitzungsbeschreibung übersetzt werden.

3 Anwendungen

Die traditionellen Telefonnetzwerke sind ausgereift und arbeiten sehr kosteneffizient. Auch wenn sich kurzfristig Einsparpotentiale durch IP-Telefonie ergeben, so ist mittelfristig jedoch zu erwarten, dass sich die Preise angleichen. Langfristig muss VoIP daher durch andere Vorteile punkten, von denen eine Auswahl nun vorgestellt wird.

3.1 IP-basierte TK-Anlage

Gerade für Unternehmen, die bereits ein umspannendes Firmendatennetz aufgebaut haben, kann es sich lohnen, eine VoIP-Lösung einzusetzen. In der Folge wird der interne Sprachverkehr über die Datenleitungen geführt, während externe Anrufe über ein VoIP-Gateway ins PSTN hinaus- bzw. aus dem PSTN ins Firmennetz hineingeleitet werden. Man spricht hier auch von Computer Telephone Integration (CTI) [Gonc99].

Für die Firma ergibt sich erhebliches Einsparpotential, denn nach einer Übergangsphase existiert nur noch ein Netz, das gewartet werden muss. Auch Kosten für die Anmietung bzw. Einrichtung von Telefonleitungen zu entfernten Firmenstandorten, die ans Datennetz angeschlossen sind, können so größtenteils eingespart werden. Least Cost Routing (LCR) stellt ein weiteres Schlagwort in diesem Bereich dar. Abhängig von den Kosten und der Netzauslastung kann die TK-Anlage entscheiden, ob das Gespräch über das Firmennetz oder über öffentliche Leitungen gehen soll.

Die Softwaregrundlage einer solchen TK-Anlage lässt weiteren Gestaltungsraum offen. Da Mitarbeiter in verschiedenen Abteilungen bzw. Dienstpositionen unterschiedliche Anforderungen an ihr Telefon stellen, können Telefone individuell konfiguriert werden. Bestimmte Zusatzdienste, wie z.B. Video-Conferencing, können je nach Bedarf zugeschaltet werden. Auch die Oberfläche des System-Telefons kann auf die jeweilige Firma zugeschnitten werden, um beispielsweise die Corporate Identity zu reflektieren.

3.2 IP-Telefon als intelligente Nachrichtenzentrale

Die Instabilität von Rechnern und Software legt nahe, ein eigenständig funktionsfähiges IP-Telefon einzusetzen, das direkt mit dem Datennetz verbunden ist. Bei ausgeschaltetem PC oder bei einem Systemabsturz bleibt es mit eingeschränkter Funktionalität betriebsfähig und kann beispielsweise auf eingegangene Emails aufmerksam machen.

In Verbindung mit einem PC kann es mit vielfältigen Funktionen ausgestattet werden und als intelligente Nachrichtenzentrale dienen. Vorstellbar ist die Anbindung an einen Zeitplaner wie MS-Outlook. Bei Abwesenheit eines Firmenmitarbeiters kann ein Anrufer nach einem Rückgriff auf den Outlook-Terminkalender informiert werden, wann der Gesprächspartner wieder verfügbar ist. Ebenso kann das Netz automatisch auf andere zuständige Mitarbeiter weiterleiten und so für effizienteres Kundenmanagement sorgen [AG00].

3.3 Web-based Call-Center

Eine Erweiterung eines herkömmlichen Call Centers stellt die bereits verbreitete Form des Web-based Call-Centers dar. In ihm sind die Kundendienstmitarbeiter sowohl mit dem Internet als auch dem Telefonnetz verbunden und können über beide Netze Sprachverbindungen herstellen. Dem Kunden wird es so möglich aus dem Internet oder sogar direkt aus einer Anwendung heraus per “Click-to-dial“-Button eine Verbindung zum Call-Center-Mitarbeiter

herzustellen und mit diesem zusammen das Problem live auf dem Rechner durchzugehen. Besitzt der Kunde keine VoIP-Lösung, hinterlässt er eine Telefonnummer und wird nach kurzer Zeit vom Call-Center auf dem Telefonnetz zurückgerufen [Gonc99].

Die Verbindung von Web, PC und Telefonie bringt weitere Vorteile mit sich. Das Anwendungsprogramm kann die Anfrage des Kunden direkt an einen kompetenten Ansprechpartner im Call-Center weiterleiten. Zeitgleich könnten dem Call-Center auch spezifische Informationen aus der Kundendatenbank zur Verfügung gestellt werden.

3.4 Groupware Systeme

Aufgrund der zunehmenden Globalisierung gewinnen Systeme für Heimarbeit und entfernte Projektarbeit immer mehr an Bedeutung. Hier kann VoIP gerade durch die Integration von Sprach- und Datenkommunikation überzeugen. Entfernte Projektteilnehmer können sich ins Firmennetzwerk einwählen und, während sie miteinander sprechen oder sogar per Video-Konferenz visuellen Kontakt haben, gleichzeitig an Dokumenten arbeiten (Document-Sharing). Auch Heimarbeiter werden nach der Einwahl ins Firmennetz unabhängig vom Standort über eine VoIP-Lösung telefonisch erreichbar.

In gleichem Maße kann IP-Telefonie auch erhebliche Vorteile beim virtuellen Lernen bringen. Der Kontakt über integriertes VoIP zwischen Lehrenden und Lernenden wird wesentlich erleichtert [Gonc99].

3.5 Unified Messaging Systeme

Bei Unified Messaging Systemen (UMS) werden Nachrichten, die bisher getrennt gespeichert wurden, unter einer Anwendung zusammengeführt. Über ein einziges Portal kann der Benutzer somit auf Anrufbeantworter, Fax und Email zugreifen. Durch den Einsatz von Datennetzbasierter Telefonie wird ermöglicht, auch Sprachnachrichten mit in diese Nachrichtenzentrale einzuflechten. Ein Firmenmitarbeiter kann sich bei einem solchen System beispielsweise von unterwegs per Handy ins Firmennetzwerk einwählen und seinen Eingang von Email-, Fax-, und Sprachnachrichten prüfen. Durch einfache Integration eines Softwaremoduls in die VoIP-Lösung kann er sich Nachrichten auch vorlesen lassen [AG00].

3.6 Call-by-Call über IP

Seit der Deregulierung des deutschen Telekommunikationsmarktes hat sich die Call-by-Call-Technik im Telefonnetz erfolgreich etabliert. Da gerade für Privatanwender eine Nutzung des Heim-PCs für Internet-Telefonie aufgrund von technischen Beschränkungen (noch) nicht attraktiv erscheint, bietet sich hier eine Alternative.

Durch Einsatz der Netzvorwahl eines entsprechenden Providers wird der Anruf zunächst auf dessen nächstgelegenen Standort umgelenkt. Dort speist der Provider das Telefongespräch ins Internet ein und überträgt es bis zu seinem Standort in der Nähe des Empfängers. Hier werden die Daten zurück ins Telefonnetz zum Angerufenen geschleust. Für beide Benutzer erfolgt dies transparent und macht sich höchstens durch verminderte Sprachqualität bemerkbar. Auf Basis dieses Prinzipes führt die Deutsche Telekom einen Feldversuch durch, bei dem die Daten über eigene Internet-Backbones geschickt werden.

4 Kritische Betrachtung

Wenn die IP-Telefonie so viele Vorteile hat, warum telefoniert dann nicht jeder über das Internet? Ein Hauptgrund liegt darin, dass sie nach heutigem Entwicklungsstand bei Qualität und Zuverlässigkeit noch nicht mit traditionellen Sprachdiensten mithalten kann. Gründe hierfür und mögliche Lösungen sollen im folgenden aufgezeigt werden.

4.1 Analyse

4.1.1 Verzögerung/Latenz

Unter Verzögerung versteht man die Laufzeit, die Sprache benötigt, um vom Sprechenden zum Gegenüber zu gelangen. Bereits eine Verzögerung von mehr als 200ms wirkt sich störend auf die Sprachqualität aus. Es können dann Überlappungen auftreten, wenn der Gegenüber zu früh zu sprechen beginnt, oder Echoeffekte [Perc99].

Grundsätzlich unterscheidet man in Telefonnetzwerken drei Arten von Verzögerung. Ausbreitungsverzögerung entsteht bei der Ausbreitung von Licht (300.000 km/s) und Elektronen (200.000 km/s) durch den Leiter. Zur Verarbeitungsverzögerung kommt es durch die Geräte, die Pakete durchs Netz leiten. Bei einem paketbasierten Netzwerk ergibt sich zusätzlich die sogenannte Aufreihungsverzögerung, die bei der Paketübergabe und beim Datenstau in Warteschlangen verursacht wird.

Nach der G.114-Empfehlung der ITU-T darf bei einer guten Sprachübertragung nicht mehr als 150ms einseitige Ende-zu-Ende-Verzögerung auftreten [DaPe00]. Wie man aus der folgenden Beispielrechnung entnehmen kann, addieren sich allerdings die Verzögerungen in der Realität vor allem bei hoher Netzauslastung schnell auf einen höheren Wert:

Verzögerungsquelle	Fest	Variabel
Kodierung G.729	5ms	
Kodierung G.729 (10ms pro Frame)	20ms	
Queuing		6ms
Aufreihung	3ms	
Ausbreitung (private Leitungen)	32ms	
Netzwerk (öffentlich)	50ms	
Jitter-Buffer		2-200ms
Gesamt (mit 50ms-Jitter-Buffer)	160ms	

Tabelle 1: Verzögerungsbudget [DaPe00]

4.1.2 Echo

Ein bekanntes Problem bei der Sprachübertragung stellt das Echo dar. Da schon ein Echo ab 25ms beim Hören der eigenen Sprache irritierend wirken kann, ist es notwendig, Echoentferner einzubauen. Bei der Erstinstallation wird dabei die Echoentfernung bestimmt und eingestellt. Später wird die Sprache dann zwischengespeichert und invertiert. Kommt ein Echo zurück, wird der invertierte Teil der Sprache auf die Frequenzen addiert und somit der Echo-Effekt eliminiert [DaPe00].

4.1.3 Jitter

In paketbasierten Netzen bezeichnet man die Zeitschwankung zwischen der Ankunft einzelner Pakete als Jitter. Um diesen auszugleichen, ist ein Jitter-Buffer nötig, der Pakete zwischenspeichert und in der richtigen zeitlichen Reihenfolge ausgibt. Da mit zunehmender Größe des Jitter-Buffers die Gesamtverzögerung verlängert wird, kommt in modernen Systemen ein dynamischer Buffer zum Einsatz, dessen Größe von der Qualität der Verbindung abhängt. Mit RTP-Zeitstempeln lässt sich dabei die Jittergröße im Netzwerk bestimmen [DaPe00].

4.1.4 Paketverlust

Da in Paketnetzwerken Paketverluste an der Tagesordnung sind, werden verlorene Daten bei normaler Datenkommunikation häufig erneut übertragen. Bei VoIP macht dies allerdings wenig Sinn, denn es handelt sich um eine Echtzeitanwendung. Nach der Empfehlung G.114 der ITU machen sich aber erst Paketverluste über 5% bei einem Gespräch als störend bemerkbar. Tritt zusätzlich ein Verzögerung unter 200ms ein, liegt nach G.114 “sehr gute“ Sprachqualitiät (Toll Quality) vor, unter 400ms immerhin noch “akzeptable“ Qualität [AG00].

4.1.5 Ausfallzeiten

Das heutige Telefonnetzwerk erreicht eine extrem hohe Ausfallsicherheit von 99,9998%, was auf Jahressicht fünf Minuten Störung entspricht. Während bei LAN-Netzen 99,8% (18 Stunden pro Jahr) machbar sind, erreichen WAN-Netze (z.B. Internet) nur 98% (eine Woche pro Jahr). Die Zuverlässigkeit der Datennetze muss also noch erheblich verbessert werden, damit Firmen, die auf VoIP-Lösungen setzen, ausreichend vor Ausfallrisiken geschützt sind.

Ein weiterer Problempunkt besteht in der Zuverlässigkeit der Clients, die vor allem durch Softwareinstabilität beeinträchtigt wird. Hier lässt sich durch den Einsatz von IP-Telefonen, die auch unabhängig vom angeschlossenen Rechner funktionieren, Abhilfe schaffen [AG00].

4.1.6 Firewalls

Firewalls stellen einen wesentlichen Baustein für die Absicherung privater Netze gegen Angriffe von aussen dar. In Verbindung mit IP-Telefonie-Programmen kommt es allerdings bei heute üblichen Systemen häufig zu Konflikten, denn sie behandeln diese wie eine “herkömmliche Anwendung“.

Ein mögliches Problem besteht darin, dass die Firewall eine Kommunikationsverbindung über die Netzgrenze hinaus verhindert. Durch das Zwischenschalten einer Datenverkehrsprüfung kann die Sprachqualität erheblich beeinträchtigt oder sogar gänzlich unmöglich werden. Schliesslich kann es zu Kompromissen hinsichtlich der Firewall-Schutzfunktion kommen, d.h. einzelne Funktionen müssen deaktiviert werden. Diese Konflikte sind in einer IP-Telefonie kompatiblen Firewall zu berücksichtigen [RoAS01].

4.2 Qualitätsverbesserung

Um langfristig die Nutzer zufriedenzustellen, müssen von VoIP Anforderungen an die *Quality of Service (QoS)* besser unterstützt werden. Aus diesem Grund wurde von der IETF das Resource Reservation Protokoll (RSVP) entwickelt. Es unterstützt QoS speziell für Multimedia-Dienste, indem Netzwerk-Bandbreite zur Garantie einer ausreichenden Übertragungsqualität vorab reserviert wird [Gonc99].

Neben der nötigen QoS-Unterstützung kommt auch dem stetigen *Ausbau der Übertragungsbandbreite* eine wesentliche Bedeutung zu. Dadurch kann der wachsende Datenverkehr besser verteilt werden und VoIP hat auch in Hochlastzeiten mit höherer Wahrscheinlichkeit ausreichend Bandbreite zur Verfügung.

Eine Möglichkeit zur Einsparung von Bandbreite stellt die *Sprachaktivitätserkennung* dar. Während in heutigen Sprachnetzwerken noch 64 kbit/s an Bandbreite reserviert werden, unabhängig davon, ob gesprochen wird oder nicht, kann man durch Voice-Activity-Detection (VAD) mindestens 50 % verschwendete Bandbreite ausnutzen. VAD steuert die Sprachübertragung, indem es nach einer bestimmten Zeit aufhört, Sprachpakete zu schicken, wenn ein Abfall in der Sprachamplitude feststellbar ist. Probleme ergeben sich bei der Feststellung, wann Sprache endet bzw. beginnt und bei der Unterscheidung zwischen Sprache und Hintergrundrauschen. Dabei werden oft Anfänge von Sätzen abgeschnitten.

Durch zusätzliche *Sprachkomprimierung* wird bei VoIP schliesslich ermöglicht, die 64kbit/s-Rate von PCM auf bis zu 5,3 kbit/s (ITU G.723.1 ACELP) zu reduzieren. Die ITU hat hierzu die G-Reihe von Algorithmen, auch *Codecs*, standardisiert [Schu01].

Eine subjektive Messung der jeweiligen Codec-Sprachqualität erfolgt durch den sogenannten Mean Opinion Score (MOS). Durch Zuhörer erfolgt eine Bewertung von 1 (schlecht) bis 5 (exzellent). Bei der Wahl des geeigneten Codecs muss dann ein Kompromiss zwischen benötigter Komprimierungsgrad/Bitrate und Sprachqualität gefunden werden [DaPe00].

5 Rechtlicher und Wirtschaftlicher Rahmen

5.1 Rechtsfragen

Weltweit herrscht Unklarheit über die rechtliche Einordnung der IP-Telefonie bei der Telekommunikationsgesetzgebung. Entscheidend hierbei ist, ob VoIP einen leitungsvermittelten Sprachtelefondienst darstellt oder nur eine paketorientierte Datenübertragung. Nur im ersten Fall sind Anbieter verpflichtet, Lizenzgebühren zu entrichten. Damit ergibt sich ein nicht zu unterschätzendes Risiko für die Anbieter, denn die Einführung von Gebühren könnte Kostenvorteile zunichte machen.

Während man sich darüber im klaren ist, dass ein “geschäftsmäßiges und gewerbliches Erbringen eines Dienstes“ vorliegt, ist weiterhin strittig, ob VoIP trotz technisch bedingter Verzögerungen als Echtzeitdienst gelten kann. Außerdem können sich die Pakete auch in Netzen bewegen, in denen das Telekommunikationsgesetz nicht gültig ist. Letztlich lässt es sich aufgrund der streckenweisen paketvermittelten Übertragung über ein Datennetz auch als einfache Datenübertragung klassifizieren [Bart00].

Auf EU-Ebene wird Internet-Telefonie bei der Gesetzgebung bis heute nicht explizit erwähnt. Sie kann nach EU-Auffassung erst konzessionspflichtig werden, wenn sie in den Bereich “Öffentlicher Sprachtelefondienst mittels eines selbst betriebenen festen Telekommunikationsnetzes“ fällt. Zusätzlich wird VoIP von der EU derzeit aufgrund unterlegener Sprachqualität bis auf weiteres von der Konzessionspflicht ausgeschlossen [ITU00].

5.2 Staatliche Regulierung

Weltweit unterschiedliche rechtliche Auffassungen schlagen sich auch in stark divergierenden regulatorischen Maßnahmen der einzelnen Staaten nieder. Diese reichen vom vollkommenen Verbot bis zu bedingungsloser Genehmigung.

Eine sehr liberale Einstellung hat sich in den USA entwickelt. Dort wird IP-Telefonie zwar als Telekommunikationsdienst bezeichnet, aber nicht als ein solcher behandelt. In Ländern mit einem stärker regulierten Telekommunikationssektor oder gar einem staatseigenen monopolistischen Betreiber, tendiert man dagegen eher zu rigiden Einschränkungen oder gar einem Verbot von VoIP (z.B. Ägypten, Thailand) [ITU00]. Die sehr unterschiedliche Regulation weltweit wirft eine Vielzahl von Fragen auf, da sich der Verkehr über Datennetze nur sehr schwer kontrollieren lässt. Folglich besteht noch viel Abstimmungsbedarf.

5.3 Der VoIP-Markt

In Deutschland ging man bis zum Zeitpunkt der Liberalisierung des Telekommunikationsmarktes im Januar 1998 von einem immensen Wachstum der IP-Telefonie aus. Allerdings kam es danach zu einem derart starken Preisrutsch bei Ferngesprächen, dass neue VoIP-Anbieter ihre Dienste bald einstellen mussten.

Heute hat man eine etwas nüchternere Meinung zum Thema VoIP erlangt. Die großen Player wie Telekom und Mannesmann Vodafone machen keine konkreten Angaben über geplante Projekte. Allerdings zeigt sich das Interesse der Telekom an ihrer Beteiligung an der israelischen Vocaltec [o.V.00].

Bei Vorhersagen über das zukünftige Marktvolumen gehen die Schätzungen stark auseinander. Das Marktforschungsinstitut IDC geht davon aus, dass sich der IP-Telefonie-Verkehr von 2,7 Mrd. Minuten im Jahr 1999 auf 135 Mrd. Minuten mit US$ 19 Mrd. im Jahr 2004 vervielfacht. Andere Institute (Deltathree.com, Tarifica, Analysys) sagen einen Marktanteil von 25-40 % bis 2004 voraus [ITU00]. Zusammenfassend lässt sich sagen, dass man weiterhin von gigantischen Wachstumsraten ausgeht.

5.4 Wirtschaftliche Auswirkungen

Die Telefonie über IP-Netze hat begonnen einen kompletten Wirtschaftszweig unter Druck zu setzen. Es wird auch kleineren Anbietern, wie z.B. Internet Service Providern, möglich, ihren Kunden Telefoniedienste anzubieten und damit in direkten Wettbewerb mit traditionellen Anbietern zu treten.

Der Wettbewerbs- und Liberalisierungsdruck auf die Telekommunikationsmärkte wird weiterhin steigen. Dabei liegt der Hauptnutzen beim Kunden, denn die Preise fallen und die Services müssen sich verbessern. Er muss allerdings einen Trade-off eingehen zwischen Sprachqualität und Kosten. Wenn die IP-Telefonie es schafft, den Kunden gute Qualität zu günstigeren Konditionen zu bieten, wird sie Telefonnetzanbietern weiterhin Marktanteile abnehmen. Als entscheidend erweist sich hierbei die Etablierung auf dem Firmenkundensektor, denn Privatanwender tragen verhältnismäßig wenig zum Marktvolumen bei.

Ein weiterer Effekt ist auf das internationale Settlement System in der Telekommunikation zu beobachten. Unter diesem System müssen Anbieter, bei denen ein Gespräch beginnt, Kompensationszahlungen an das Land, in dem das Gespräch endet, leisten. Die Abrechnung erfolgt durch Gegenüberstellung des Sprachverkehrs zwischen zwei Ländern auf Basis vereinbarter Preise. Da durch Umleiten des Telefonverkehrs auf IP Backbones diese Kosten umgangen werden können, wird mehr und mehr Verkehr umgeleitet [ITU00].

6 Ausblick

IP-Telefonie befindet sich auf dem besten Weg von der “Spielerei“ auf dem Heim-PC zu einer echten Alternative zu traditionellen Telefonnetzen. Nicht zuletzt die umfangreichen Aktivitäten grosser Hersteller wie Siemens, Lucent, Cisco, um nur einige zu nennen, zeugen davon.

Bis VoIP allerdings mit Sprachnetzen wirklich konkurrieren kann, besteht allerdings noch erheblicher Verbesserungsbedarf, vor allem was die Garantie der Servicequalität und den Ausbau der weltweiten Bandbreiten betrifft. Für Firmen überwiegt heute noch vielfach das Ausfallrisiko gegenüber den möglichen Kosteneinsparungen. Des weiteren müssen die Standardisierungsarbeiten weiter forciert und die Interoperabilität verbessert werden.

Nach der Klärung regulatorischer und rechtlicher Fragestellungen hat VoIP das Potential langfristig einen erheblichen Anteil des Sprachverkehrs zu übernehmen. Diesen Trend belegen aktuelle Prognosen vieler Marktforschungsinstitute (Abschnitt 5.3), die von einem immensen Wachstum ausgehen.

Ausserdem kommt man der erwarteten Konvergenz multimedialer Ströme, wie Sprache, Video und Daten einen grossen Schritt näher. Die Vision eines einzigen weltumspannenden IP-Netzes, über das sowohl Daten- als auch Sprachverkehr abgewickelt werden, könnte in nicht allzu ferner Zukunft Realität werden.

Literatur

[AG00] SWYX Communications AG. Leitfaden IP-Telefonie. *http://www.swyx.de/basics*, Juli 2000.

[Bart00] Matthias Barth. Internet-Telefonie: Heutiger Status und zukünftige Potentiale. *Bundesministerium für Bildung und Forschung*, 2000.

[DaPe00] Jonathan Davidson und James Peters. *Voice over IP Grundlagen.* Markt+Technik, München. 2000.

[FKSS01] Stefan Foeckel, Jiri Kuthan, Dorgham Sisalem und Ilona Schubert. OSIP: Eine Open Source SIP Architektur. *Praxis der Informationsverarbeitung und Kommunikation* 1(01), Januar 2001, S. 18–23.

[Gonc99] Marcus Goncalves. *Voice over IP Networks.* McGraw-Hill, New York. 1999.

[ITU00] ITU. Background Issues Paper. *IP Telephony Workshop (Genf)*, Juni 2000.

[MiMi98] Daniel Minoli und Emma Minoli. *Delivering Voice over IP Networks.* John Wiley & Sons, New York. 1998.

[Perc99] Alan Percy. Understanding Latency in IP Telephony. *http://www.brooktrout.com/whitepaper/iptel_latency.htm*, Februar 1999.

[RoAS01] Utz Roedig, Ralf Ackermann und Ralf Steinmetz. IP-Telefonie und Firewalls, Probleme und Lösungen. *Praxis der Informationsverarbeitung und Kommunikation* 1(01), Januar 2001, S. 32–40.

[Schu01] Henning Schulzrinne. Internet Telefonie - Mehr als nur ein Telefon mit Paketvermittlung. *Praxis der Informationsverarbeitung und Kommunikation* 1(01), Januar 2001, S. 4–12.

[SiSc00] Kundah Singh und Henning Schulzrinne. Interworking between SIP/SDP and H.323. *Proceedings of the 1st IP-Telephony Workshop (IPtel 2000), (Berlin, Germany)*, April 2000.

[Stüt01] Heinrich Stüttgen. Internet Telefonie - Brauchen wir sie wirklich. *Praxis der Informationsverarbeitung und Kommunikation* 1(01), Januar 2001, S. 2–3.

Abbildungsverzeichnis

Routing in Ad-Hoc-Netzwerken

Sabine Wendhack

Kurzfassung

Ad-hoc-Netzwerke sind Netze, die spontan aufgebaut werden und somit weder über eine Backbonestruktur noch über eine zentrale Verwaltung verfügen. Sie zeichnen sich dadurch aus, dass Teilnehmer ständig ein- bzw. austreten sowie ihren Standort innerhalb des Netzwerkes wechseln. Bedingt durch diese Eigenschaften stellen sie, insbesondere was Routing anbelangt, andere Anforderungen als infrastrukturbasierte Netze. Im folgenden Text wird zunächst kurz auf die Probleme von Ad-Hoc-Netzwerken im Vergleich zu konventionellen Netzwerken eingegangen. Anschließend werden generelle Charakteristika von Routingprotokollen erwähnt. Schließlich werden einige der wichtigeren Routingprotokolle erläutert und kurz verglichen.

1 Einführung

Ein Ad-Hoc-Netzwerk besteht aus Geräten, die willkürlich und ohne eine feste Backbonestruktur oder eine zentrale Verwaltung ein Netzwerk bilden. Ad-Hoc- Netzwerktechnologie wird wegen ihrer sinkenden Kosten und dem immer größeren Bedarf an Mobilität immer wichtiger.

Ad-Hoc-Netzwerktechnologie wird in militärischen Bereichen sowie bei Katastrophen bzw. Notfällen verwendet, wenn es notwendig ist, unter geografisch schwierigen Bedingungen Informationstechnologie schnell bereitzustellen. Außer in diesen Bereichen existiert ein hohes Potential für die kommerzielle Nutzung. Sofortige Netzwerkbildung während Meetings oder Konferenzen, um Information zu nutzen, ohne eine zentrale Netzwerkverwaltung aufbauen zu muessen, wird in Zukunft immer wichtiger werden.

Netzwerktechnologie so wie sie für feste Netze benutzt wird, kann nicht ohne größere Anpassungen auf Ad-Hoc-Netze übertragen werden. Letztere besitzen Eigenschaften, die andere Lösungen verlangen. Dies ist der Fall besonders in Bezug auf Routingmechanismen.

Bevor ich Ad-Hoc-Netzwerke und Routing im Allgemeinen, Unterschiede zwischen Ad-Hoc- und festen Netzen und die Herausforderung, die Ad-Hoc-Netzwerke an Routing stellen betrachte, möchte ich einige Termini definieren, die ich in dieser Arbeit verwenden werde.

Jedes Gerät, das mit anderen Geräten verbunden ist, wird *Knoten* genannt. Die direkte Verbindung zweier Knoten heißt *Link*. Auf diskrete Informationsblöcke, die über diese Links übermittelt werden werde ich mich mit *Paketen* beziehen. Die Links, die ein Paket von der Quelle zum Ziel zurücklegt wird sein *Pfad* genannt. Wenn ein Paket von einem Knoten zu einem Nachbarn übermittelt wird, wird gesagt, es habe einen *Hop* zurückgelegt. Das System der Links in einem Netzwerk wird als seine *Topologie* bezeichnet.

Ad-Hoc-Netzwerke verbinden mobile Geräte zeitweise. Während in konventionellen Netzwerken jeder Knoten seine ständige Adresse hat und die Netzwerktopologie sich nur in Ausnahmefällen ändert, sind die Knoten von Ad-Hoc-Netzwerken dynamisch in willkürlicher Weise

verbunden. Im Gegensatz dazu arbeiten herkömmliche Mobilnetze mit festen Basisstationen, über die sich die mobilen Knoten miteinander in Verbindung setzen.

Um in der Lage zu sein, ein Paket zu seinem Ziel zu leiten benötigen die Knoten, die den Pfad des Pakets bilden, Information über die Topologie des Netzwerks oder zumindest Mechanismen, um diese Information zu erhalten. Der Prozeß, ein Paket durch das Netz zu seinem Ziel zu leiten wird *Routing* genannt, die Art und Weise und die Mechanismen, die dazu benutzt werden sind durch das verwendete Routingprotokoll festgelegt.

Die Eigenschaften eines Ad-Hoc-Netzwerkes bedeuten grundsätzliche Unterschiede im Vergleich zu einem konventionellen Netzwerk. Der hauptsächliche Unterschied ist, dass es keine zentrale Backbonestruktur gibt, was bedeutet dass dem Ad-Hoc-Netzwerk ebenso keine zentrale Verwaltungseinheit zur Verfügung steht. Folglich gibt es die traditionellen Rollen von Client, Server und Router hier nicht mehr klar voneinander abgegrenzt und Routing wird von allen Knoten durchgeführt. Zusammen mit der sich dauernd ändernden Netzwerktopologie erfordert dies andere Technologien, besonders was den Routing-Prozeß betrifft.

Wenn man über Ad-Hoc-Netzwerke spricht, meint man im Allgemeinen drahtlose Ad-Hoc-Netzwerke. Theoretisch schließen Ad-Hoc-Netzwerke ebenso verdrahtete Netze, die ihre Topologie ständig ändern, ein. Diese Systeme haben jedoch in der Praxis wenig Bedeutung.

Im folgenden Text werde ich kurz die Unterschiede zwischen Ad-Hoc- und festen Netzen ansprechen, um die Aspekte herauszustellen, die andersartige Lösungen erfordern. Dann werde ich kurz generelle Unterschiede in der Konzeption von Routingprotokollen betrachten, um anschließend kurz einige wichtige Routingprotokolle zu besprechen und zu vergleichen.

2 Herausforderungen an Routing in Ad-Hoc-Netzwerken

2.1 Mobilität

Der gravierendste Unterschied von Ad-hoc-Netzwerken sowohl im Vergleich zu konventionellen verdrahteten als auch zu drahtlosen infrastrukturbasierten Netzwerken ist die Mobilität aller teilnehmender Knoten. Eine zentrale Backbonestruktur ist in dieser Art von Netzwerk nicht vorgesehen. Alle Geräte können sich innerhalb des Netzes bewegen oder auch herausfallen. Neue Teilnehmer können hinzukommen. Das bedeutet eine sehr häufige Veränderung der Netzwerktopologie.

Die Änderung der Netzwerktopologie ist etwas, das auch konventionelle Routingprotokolle bewältigen müssen. Der Prozeß, nach einer Topologieänderung wieder in einen stabilen Zustand zurückzukehren wird als *Konvergenz* bezeichnet, die Zeit die dafür benötigt wird als *Konvergenzzeit*.

Während in festen Netzen eine Topologieänderung selten vorkommt, ändert sich die Topologie in Ad-Hoc-Netzwerken ständig. Folglich ist hier vom Routingprotokoll eine sehr viel größere Anpassungsfähigkeit gefordert.

Auch ein Ad-Hoc-Netzwerk muß jedoch eine bestimmte Stabilität in Bezug auf seine Topologie aufweisen, um ein Routingprotokoll benutzen zu können. Das entgegengesetzte Extrem eines festen Netzwerkes, ein absolut instabiles Ad-Hoc-Netzwerk kann Routing nur durch ständiges *Fluten* durch das gesamte Netzwerk vollführen [Lesi98].

2.2 Energie

Ad-Hoc-Netzwerke sind im Allgemeinen drahtlose Netzwerke, das heißt, sie arbeiten mit der begrenzten Menge an Energie einer Batterie. Deswegen ist Stromverbrauch für sie ein kri-

tischer Aspekt, der so effizient wie möglich gestaltet werden muß. Das betrifft auch den Energieverbrauch des Routingprozesses.

Routingprotokolle unterscheiden sich im Energieverbrauch. Dies variiert mit der Anzahl der Schritte, die sie vollziehen, mit der Anzahl der Nachrichten, die sie senden und der Häufigkeit, mit der dies nötig ist.

Außerdem gibt es in einem Netzwerk mehr oder weniger energieintensive Pfade, die benutzt werden können. Ein Routingprotokoll muß die Wahl eines Pfades in Bezug auf Energie optimieren. Wie in [ChTa00] gesagt wird, ist es wichtig, sich nicht auf die Optimierung des gesamten Energieverbrauchs im Netzwerk zu fixieren. Knoten eines energieeffizienten Pfades würden dann schnell ihre Energiereserven aufbrauchen, was zu einer Netzwerkteilung führen könnte, obwohl es andere Pfade gegeben hätte, die insgesamt mehr Energie verbraucht aber eine Netzwerkteilung noch verhindert hätten. „It turns out that in order to maximize the lifetime [of the network], the traffic should be routed such that the energy consumption is balanced among the nodes in proportion to their energy reserves, instead of routing to minimize the absolute consumed power.“ [ChTa00]

2.3 Bandbreite

Ein anderer kritischer Aspekt, den Routingprotokolle beachten müssen, ist die benötigte Bandbreite. Die meisten Ad-Hoc-Netzwerke arbeiten drahtlos, was aufgrund von technischen Bedingungen weniger verfügbare Bandbreite bedeutet.

2.4 Sicherheit

Während Sicherheitsaspekte in Bezug auf konventionelle Computernetzwerke in letzter Zeit mehr und mehr diskutiert wurden, steckt die Diskussion mit Bezug auf Ad-Hoc-Netzwerke noch in ihren Anfängen. [Kärp00]

Ad-Hoc-Netzwerke sind unter dem Sicherheitsaspekt anfälliger als konventionelle Netzwerke. Die Teilnehmer wechseln beständig,was eine wesentlich schmälere Basis an Vertrauen zwischen den Knoten zur Folge hat. Bedingt durch die Technik kann drahtlose Kommunikation weit einfacher angegriffen werden, beziehungsweise ausfallen, als verdrahtete.

Routingprotokolle, die an Ad-Hoc-Technologie angepasst wurden und andere Aspekte wie die Mobilität gut bewältigen, bewältigen den Sicherheitsaspekt in der Regel nicht genügend gut. Die wenigen Sicherheitsmechanismen, die sie anbieten sind im Nachhinein in existierende Routingprotokolle eingebaut worden, was zusätzliche Probleme verursachen kann. [Kärp00]

3 Routingprotokolle

3.1 Allgemeines

3.1.1 „Table-Driven“ versus „On-Demand“

Routingprotokolle für Ad-Hoc-Netzwerke werden, wie die für konventionelle Netze in table-driven- (proaktive) und on-demand- (reaktive) Protokolle unterteilt. Table-driven-Protokolle gründen auf aktueller Routinginformation, die in jedem Knoten in einer sogenannten Routingtabelle ständig vorhanden sind. Sie reagieren auf Topologieänderungen indem sie Updates verschicken um eine konsistente Information über die Topologie im Netzwerk zu erhalten. [RoTo00]

Der Vorteil dieser Vorgehensweise ist, dass wenn Routinginformation vorhanden ist, ihre Benutzung in effizienter Weise geschieht. Die rein proaktiven Protokolle sind wegen des ständigen intensiven Austauschs von Routinginformation, den sie erfordern für Ad-Hoc-Netzwerke allerdings weniger geeignet. [Kärp00]

On-demand-Protokolle dagegen erstellen Routinginformation nur, wenn dies vom Quellknoten gewünscht wird. Routinginformation wird mittels des „Route discovery"-Prozeß festgestellt. Wenn ein Pfad festgestellt ist, wird er gepflegt bis er nicht mehr gültig oder nicht mehr erwünscht ist.

Dadurch wird genereller Netzwerkverkehr gering gehalten, während ein bestimmter Overhead entsteht, wenn ein Pfad gebraucht wird, weil dieser erst gefunden werden muß. Wegen ihres niedrigen Netzwerkverkehrs und der damit reduzierten Bandbreiten- und Energieauslastung, was ein Argument in Ad-Hoc-Netzwerken ist, benutzen viele Ad-Hoc-Netzwerkprotokolle einen on-demand-Ansatz. Außerdem gibt es die Möglichkeit, hybride Protokolle zu verwenden, welche beide Ansätze verwenden.

3.1.2 Flache versus hierarchische Routingphilosophie

Viele der älteren Routingprotokolle betrachten alle Knoten, als ob sie auf demselben Niveau arbeiten. Das heißt, es gibt keine Knoten, die anderen irgendwie übergeordnet wären, sei es zum Beispiel indem sie generell mehr Routinginformation erhalten oder einen größeren Teil der Routingfunktionen übernehmen. Diese Protokolle arbeiten auf Basis einer flachen Adressierung.

Es gibt verschiedene Ansätze, um Routing basiert auf hierarchischer Routingphilosophie durchzuführen. Das heißt, dass nicht alle Knoten auf demselben Niveau arbeiten, sondern einige, entweder ständig oder bezüglich eines bestimmten Pfades, einen Sonderstatus haben.

Es gibt die Möglichkeit, in einem Netzwerk virtuelle Backbones zu generieren, die zentrale Funktionen übernehmen. Mit CGSR, einem Routingprotokoll, das später besprochen wird, wird sämtliches Routing durch den jeweiligen *Clusterhead* des Subnets geleitet, das sind Knoten, die eine zentrale Position im Netzwerk übernehmen. Es gibt andere Ansätze, so zum Beispiel beschrieben von [LiHa00], die virtuelle Backboneknoten nur als Speicher für Adressen einführen, während Routing flach durchgeführt wird. Die Herausforderungen in Bezug auf diese Ansätze sind die Heuristiken, mit deren Hilfe in der sich ständig verändernden Netzwerktopologie die übergeordneten Knoten dynamisch bestimmt werden können. Für diesen Prozeß sollen möglichst wenig Ressourcen gebraucht werden, aber trotzdem gesichert werden, dass alle Knoten eingebunden und der Knoten gewählt wird um die Funktion des virtuellen Backbone bzw. des Clusterhead wahrzunehmen, der in Relation zu den anderen optimal gelegen ist.

Ein anderer Ansatz ist der, ein hierarchisches Adressschema zu benutzen, ohne ausgewählte Knoten als virtuelle Backbones oder Clusterheads zu bestimmen. Das Virtual Subnet Protocol (VS) unterteilt zum Beispiel ein größeres Netzwerk in kleinere logische Einheiten, sogenannte *Subnets*. Adressen in solch einem Netzwerk bestehen aus zwei Elementen, die sich auf zwei verschiedene Niveaus beziehen: physikalische und virtuelle Subnets. Physikalische Subnets bestehen aus den Knoten, die geographisch nahe beieinanderliegen. Ein virtuelles Subnet verbindet jeweils einen Knoten eines physikalischen Subnets mit einem aus anderen physikalischen Subnets. Routing wird in zwei Etappen durchgeführt. Zuerst wird das Paket innerhalb seines physikalischen Subnets zum Knoten geleitet, der demselben virtuellen Subnet wie der Zielknoten angehört. Anschließend wird das Paket innerhalb dieses virtuellen Subnets zu seinem Ziel geleitet. [Lesi98]

Während flache Adressierung und Routingsysteme weniger kompliziert und einfacher zu benutzen sind, können hierarchische Ansätze in größeren Netzen Ressourcen einsparen.

3.2 Table-driven-Protokolle

3.2.1 Destination-Sequenced Distance-Vector Routingprotokoll

Das Destination-Sequenced Distance-Vector Routingprotokoll (DSDV) ist eine der früheren Anpassungen eines Routingprotokolls an Ad-Hoc-Netzwerke. Es benutzt ausschließlich bidirektionale Links.

DSDV benutzt Routingtabellen, die in jedem Knoten des Ad-Hoc-Netzwerks gespeichert werden. Diese Tabellen beinhalten die Adresse eines jeden Knoten des Netzwerks, zusammen mit dem nächsten Hop, den ein Paket nehmen muß, um den jeweiligen Knoten zu erreichen. Ein Knoten, der ein Datenpaket erhält, sieht demnach zuerst die Zieladresse nach und anschließend den Nachbarn, der angegeben ist, um das Paket schließlich weiterzuleiten.

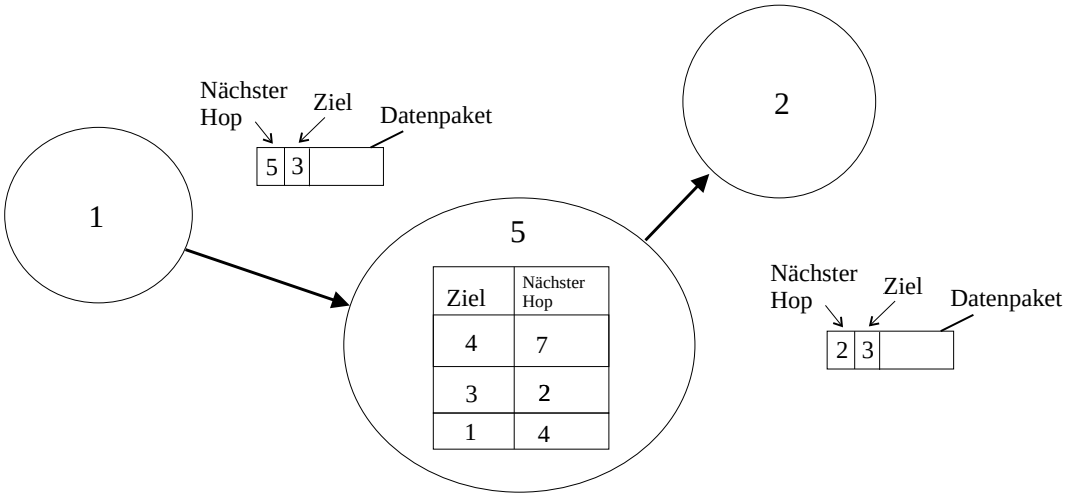

Abbildung 1: Routing nach Knoten 3 über 5 und 2 mit DSDV

Während Routing mit DSDV vergleichsweise einfach ist, stellt die Erstellung und die Pflege der Routingtabellen die eigentliche Herausforderung dieses Protokolls dar.

Um die Routinginformation in jedem der Knoten zu aktualisieren, enthalten die Routingtabellen zusätzlich zur Zieladresse und der Angabe des nächsten Hop ein „Route metric“ und eine Folgenummer. Das „Route metric“ wird von jedem Knoten, den das Paket durchläuft erhöht, so dass es am Ende die Entfernung von der Quelle zum Ziel angibt.

Topologieänderungen werden entweder von der Hardware entdeckt, oder von Knoten, die keine Nachrichten von früheren Nachbarn mehr erhalten. Sobald Topologieänderungen entdeckt werden sendet der entsprechende Knoten ein „Routing table update“-Paket. Dieses Paket startet mit einer Metrik von eins. Das bedeutet jedem Nachbarn, der es bekommt, dass er einen Hop vom sendenden Knoten entfernt ist. Die Nachbarn erhöhen wiederum die Metrik und senden das Paket weiter. Dieser Prozeß wird wiederholt bis jeder Knoten im Netzwerk eine Kopie des Update-Pakets mit der jeweiligen Metrik bekommen hat. Wenn ein Knoten doppelte Update-Pakete erhält, berücksichtigt er nur dasjenige mit der kleinsten Metrik und verwirft den Rest [Lesi98]. Wenn ein ausgefallener Link entdeckt wird, wird dies vom jeweiligen Knoten durch ein Update-Paket der Routingtabellen bekanntgemacht, indem er für das entsprechende Ziel eine Metrik von unendlich angibt. Die Routingtabelleneinträge, die diesen Link benutzen werden danach von allen Knoten, die die Information erhalten, gelöscht.

Um ältere Information von aktueller zu unterscheiden, bezeichnet ein Knoten jedes Update-Paket seiner Routingtabelle mit einer Folgenummer. Ein Knoten, der solch ein Paket erhält, vergleicht demnach die Folgenummer dieses Pakets mit der der Information, die er bezüglich

der jeweiligen Zieladresse bereits in seiner Routingtabelle hat und übernimmt diejenige mit der höheren Folgenummer.

Wenn ein Routinginformations-Update-Paket weitergeleitet wird, enthält es auch die Adresse des weiterleitenden Knoten. Ein Knoten, der ein solches Paket erhält trägt diese Adresse als nächsten Hop bezüglich der aktualisierenden Knoten ein.

Das Hauptproblem von DSDV ist der Overhead, der duch die ständige Aktualisierung der Routinginformation innerhalb des gesamten Netzwerkes entsteht. Obwohl es die Möglichkeit bietet, ein Paket, das nur einen Teil der Information anstelle eines „full dump“ enthält zu versenden, wird die Größe der Routingtabellen und die erforderliche Bandbreite, um sie auf dem aktuellen Stand zu halten, mit der Anzahl der im Netzwerk verbundenen Knoten immer größer [Lesi98]. DSDV ist daher in relativ kleinen Netzwerken anwendbar, während der Verkehr, der durch die ständige Aktualisierung von Routinginformation entsteht, es in größere Netzwerken ineffizient werden läßt.

3.2.2 Clusterhead Gateway Switch Routing

Im Gegensatz zu den meisten anderen Protokollen basiert Clusterhead Gateway Switch Routing (CGSR) auf einem hierarchischen Organisationsschema. CGSR benutzt DSDV als zugrundeliegendes Konzept. Anstelle von flachem Routing wird Routing mit Hilfe von Clusterheads und Gateway-Knoten durchgeführt. Clusterheads sind Knoten, die innerhalb ihres Subnets eine zentrale Position in Bezug auf Routing einnehmen. Gateway-Knoten liegen in der Reichweite zweier oder mehr Clusterheads.

Routing wird zunächst von der Quelle zum Clusterhead des jeweiligen Subnets durchgeführt. Danach wird das Paket über Gateway-Knoten und andere Clusterheads bis zum Clusterhead des Subnets der Zieladresse weitergeleitet. Dieser Clusterhead übermittelt das Datenpaket dann zu seinem Ziel.

Um dieses Protokoll benutzen zu können, muß jeder Knoten eine Cluster-Mitglieds-Tabelle pflegen. Diese Cluster-Mitglieds-Tabellen werden mittels des DSDV-Algorithmus aktualisiert. Zusätzlich erhält jeder Knoten eine Routingtabelle, um den jeweiligen nächsten Hop zum Ziel zu bestimmen. Wenn ein Knoten ein Paket erhält, sieht er zunächst in seiner Cluster-Mitglieds-Tabelle sowie seiner Routingtabelle nach, um den nächsten Clusterhead auf dem Weg zum Ziel zu bestimmen. Dann wird anhand der Routingtabelle der zu diesem Clusterhead führende Knoten. [RoTo00].

Mit diesem hierarchischen Ansatz wird Routing effizienter. Der Nachteil dieser Form des hierarchischen Routing ist, dass die Wahl der Clusterheads Ressourcen verbraucht. Eine häufige Wahl neuer Clusterheads sollte daher durch einen „Least Cluster Change“-Algorithmus, der eine Neuwahl des Clusterhead nur dann provoziert, wenn ein Knoten den Kontakt zu sämtlichen existierenden Clusterheads verliert und nicht jedes Mal wenn Clustermitglieder wechseln, vermieden werden [RoTo00].

3.2.3 Das Wireless Routing Protokoll

Das Wireless Routing Protokoll (WRP) ist ein auf Tabellen basierendes Protokoll, das sämtliche Routinginformation in allen Knoten des Netzwerks bereitzuhalten versucht. Jeder Knoten des Netzwerks hat vier Tabellen zu pflegen: Eine „Distance“-Tabelle, eine Routingtabelle, eine „Link-cost“-Tabelle und eine „Message retransmission list“-Tabelle (MRL).

Die „Distance“-Tabelle enthält die Entfernung eines Knotens zu jedem Ziel über jeden Nachbarn, über den es möglich ist das entsprechende Ziel zu erreichen. Zusätzlich speichert es

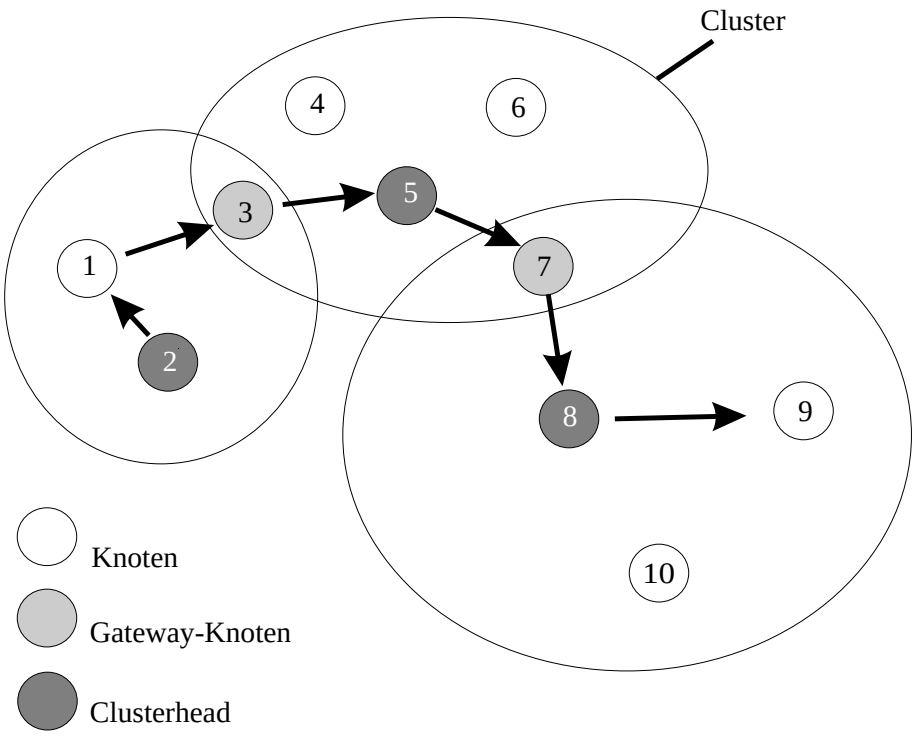

Abbildung 2: Routing mit CGSR: Von Knoten 2 zu Knoten 9

den nachfolgenden Hop jeder Möglichkeit. Die Routingtabelle enthält Information wie die Entfernung zu jedem Ziel und den Vorgänger sowie den Nachfolger des speichernden Knoten auf dem jeweiligen Pfad. Die „Link-cost"-Tabelle enthält die Kosten aller Links zu sämtlichen Nachbarn. Die MRL ist dafür verantwortlich, zu verfolgen, welche Updates wiederholt gesendet werden müssen und welche Nachbarn welche Updates bestätigen müssen.

Knoten senden Update-Nachrichten um sich gegenseitig über Topologieänderungen zu informieren. Update-Nachrichten werden nur zwischen benachbarten Knoten versandt, sie enthalten eine Liste von Updates neben einer Liste, die angibt, welche Knoten das Update bestätigen sollten. [RoTo00]

Falls ein Knoten keine Nachrichten sendet, muß er innerhalb einer bestimmten Zeit eine hello-Nachricht senden, um seine Einbindung in das Netzwerk sicherzustellen. Wenn nicht, bedeutet das Fehlen von Nachrichten einen unterbrochenen Link. [RoTo00]

Indem WRP neben der Entfernung den Second-to-last Hop für jedes Ziel speichert, vermeidet es Routing-Loops auf originelle Weise. Indem es Knoten die Konsistenz ihrer Information mit ihren Nachbarn prüfen lässt, vermeidet es das „Count-to-infinity"-Problem. Dies stellt eine schnellere Konvergenz nach der Unterbrechung von Links sicher. [RoTo00]

3.3 On-demand Protokolle

3.3.1 Dynamic Source Routing

Dynamic Source Routing stellt Routinginformation nur her, wenn sie gebraucht wird. Es benutzt Source Routing, was bedeutet dass der Sender den gesamten Pfad zum Ziel kennt. Jedes Paket enthält sämtliche notwendige Routinginformation, also eine geordnete Liste aller Knoten, die es auf seinem Pfad durchlaufen muß, in seinem Header. Pfade werden in „Route caches" gespeichert. Mit DSR hat nicht jeder Knoten sämtliche Information, die nötig ist um

Pakete weiterzuschicken. Ein Knoten sieht, wenn er ein Paket erhält einfach den nächsten Hop im Header des Pakets nach. Auf diese Weise wird Overhead reduziert, denn periodische Routingtabellenupdates sind nicht notwendig. DSR kann unidirektionale Links benutzen.

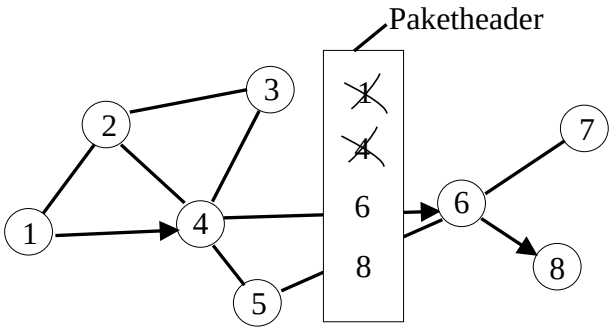

Abbildung 3: Der Routingprozess mit DSR von Knoten 1 nach 8

Während Routing mit DSR eher einfach ist, stellen „Route discovery“ und „Route maintenance“ die größere Herausforderung dar. Wenn ein Knoten einen Pfad finden möchte, flutet er das Netzwerk mit einem „Route request“-Paket. Dieses Paket enthält eine Liste der Knoten, die durchlaufen werden müssen, in die die Quelle als erste ihre eigene Adresse einträgt. Falls sie nicht das Ziel sind, über keinen Pfad zum Ziel verfügen und noch nicht auf der Liste stehen, fügen Knoten, die „Route request“-Pakete erhalten ihre Adresse hinzu und versenden sie weiter. Wenn ein Knoten die Kopie eines „Route request“-Paketes erhält, das er bereits bekommen hatte, verwirft er es.

Wenn ein Knoten das Ziel eines „Route request“ ist, oder wenn er einen Pfad zum Ziel in seinem „Cache“ findet, leitet er den „Route request“ nicht weiter sondern antwortet mit einem „Route reply“ zurück zur Quelle. Dieser „Route reply“ enthält sämtliche Routinginformation zum Ziel.

Knoten unterhalten ein „Route cache“. Sie halten hier sämtliche Routinginformation, die sie mittels vorbeikommender „Route requests“ erhalten, fest. Wenn ein Knoten ein Paket senden möchte, sieht er in seinem „Route cache“ nach, ob er über einen entsprechenden Pfad verfügt. Nur wenn er keinen findet, initiiert er einen neuen „Route discovery“-Prozeß.

„Route maintenance“ wird folgendermaßen durchgeführt: Wenn ein Knoten entlang des Pfades eines Pakets einen Fehler entdeckt, sendet er ein „Route error“-Paket zum Sender zurück. Dieses enthält die Adressen der Knoten an beiden Enden des fehlerhaften Hops. Wenn ein „Route error“-Paket erhalten oder überhört wird, wird der fehlerhafte Hop aus sämtlichen „Route caches“ gelöscht [Lesi98].

Mit DSR entsteht Overhead, wenn neue Pfade gefunden oder Topologieänderungen bewältigt werden müssen. „However, this overhead can be reduced by employing intelligent caching techniques in each node, at the expense of memory and CPU resources.“ [Lesi98]

Zusätzlich entsteht Overhead in Bezug auf Bandbreite, da die gesamte nötige Routinginformation im Header eines jeden Paketes gespeichert wird.

3.3.2 Temporally-Ordered Routing Algorithm

Der Temporally-Ordered Routing Algorithm (TORA) wurde entwickelt um Overhead von Kommunikation zu vermeiden. Optimales Routing wird als sekundär betrachtet, so dass oft längere Pfade benutzt werden um zu vermeiden, dass durch das Suchen eines besseren zusätzlicher Datenverkehr entsteht. Routinginformation wird, da es ein On-demand-Protokoll ist,

nicht ständig für alle Ziele gepflegt, sondern ausschließlich wenn sie gebraucht wird. TORA ist „source-initiated“ und kann über mehrere Pfade zu einem Ziel verfügen [RoTo00].

TORA läuft in jedem Knoten separat für jedes Ziel. In jedem Knoten errechnet es eine *Höhe* in Bezug auf jedes Ziel. Information fließt über den Knoten mit der geringstmöglichen Höhe. Wenn ein Link zwischen zwei Knoten unterbricht, setzen diese ihre Höhen auf ein lokales Maximum und versenden ein Update-Paket. Information wird dann einen alternativen Pfad finden.

Der Parameter *Höhe* hängt von der Zeit ab. Deswegen braucht TORA die synchronisierte Zeit an allen Knoten. Alle Knoten müssen daher Zugriff auf einen externen Zeitgeber haben (zum Beispiel auf GPS).

Wenn ein Knoten einen Pfad zu einem bestimmten Ziel benötigt, versendet er ein „Route query“-Paket, das die Adresse des Ziels enthält. Dieses Paket wird durchs Netz gesendet, bis es entweder das Ziel erreicht, oder einen Knoten, der einen Pfad zum Ziel besitzt. Der Empfänger des „Query“-Pakets versendet dann ein Update-Paket, dass seine Höhe in Bezug auf das Ziel enthält. Ist der Empfänger das Ziel, so ist seine Höhe gleich null. Wenn das Paket durch das Netzwerk zurückläuft, setzt jeder Knoten, der es bekommt, seine Höhe einen Wert größer als die Höhe des Nachbarn, von dem das Paket kam. Folglich werden eine Reihe von Pfaden zu einem Ziel erstellt, die durch einen geordneten Graphen abgebildet werden können.

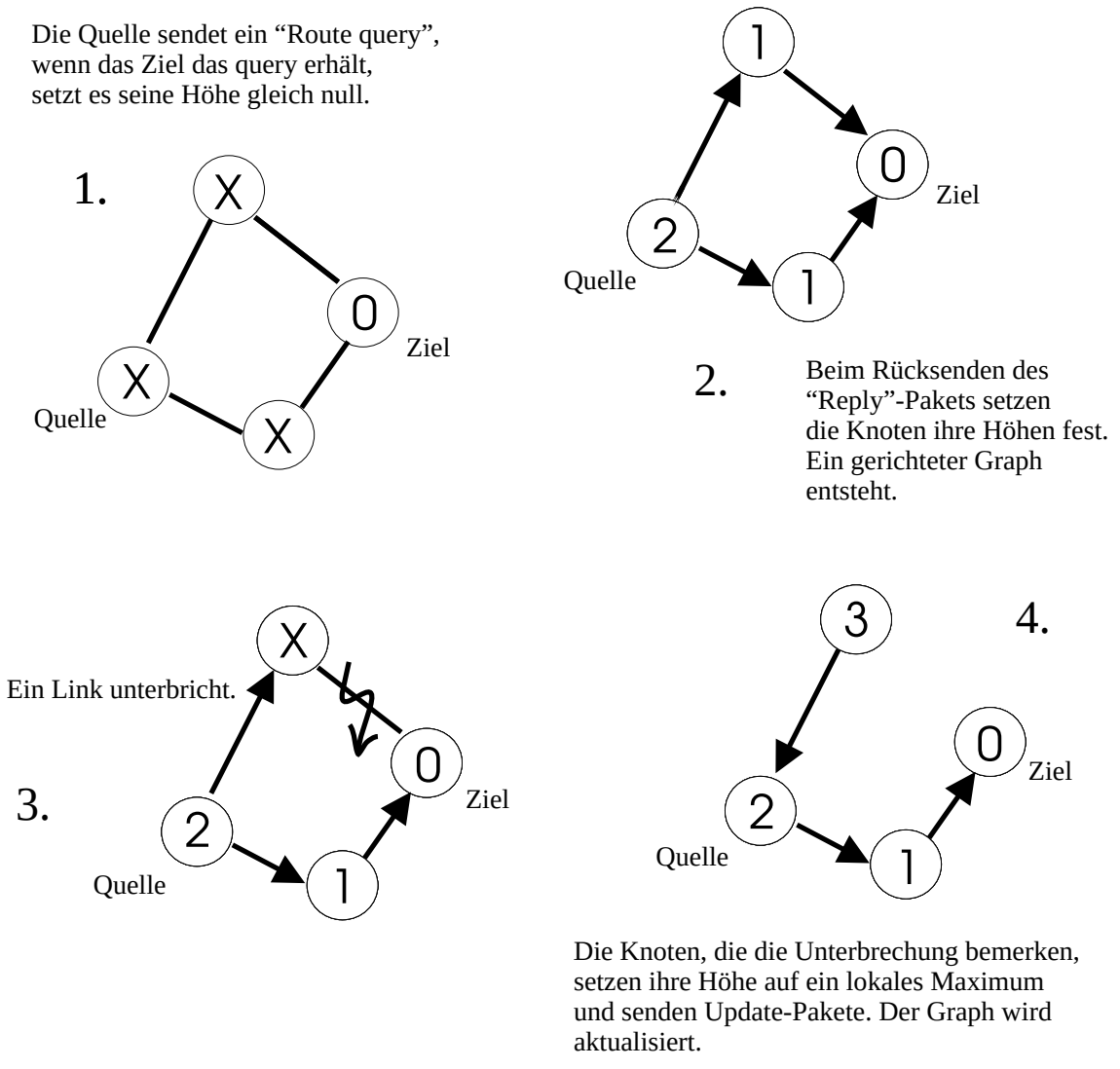

Abbildung 4: „Route discovery“ und „Route maintenance“ mit TORA

TORA ist eines der komplizierteren Routing-Protokollen. Jeder Knoten muß eine Höhe, daneben aber auch alle vorhandenen Links bezüglich jeder Verbindung, die vom Netzwerk unterstützt wird, festhalten. Außerdem muß er eine ständige Verbindung zu seinen Nachbarn halten, um Topologieänderungen zu bemerken. Auf der einen Seite benötigt TORA dadurch Speicherplatz, auf der anderen Seite CPU und Bandbreite.

Ähnlich DSDV kann TORA während der Konvergenz Routing-Loops provozieren.

3.3.3 Ad-Hoc On-Demand Distance Vector Routing

Das Ad-Hoc On-demand Distance Vector Routingprotokoll (AODV) ist eine Verbesserung von DSDV. Im Gegensatz zu DSDV minimiert AODV, indem es Pfade on-demand bereitstellt, den erforderlichen Datenverkehr.

AODV basiert im Gegensatz zu DSR nicht auf Source-Routing. Es verwendet traditionelle Routingtabellen, die nicht mehr als einen Pfad für jedes Ziel enthalten. Knoten, die nicht auf dem entsprechenden Pfad liegen, haben keinen Zugriff auf die ausgetauschte Routinginformation.

Wenn ein Knoten ein Paket zu einem bestimmten Ziel senden möchte, sendet es ein„Route request“-Paket. Jeder Knoten, der ein solches Paket erhält, versendet es weiter zu seinen Nachbarn. Wenn das Paket das gewünschte Ziel oder einen Knoten, der über einen aktuellen Pfad verfügt, erreicht, sendet der jeweilige Knoten ein „Route-reply“-Paket. Ein Knoten verwirft ein Paket, von dem er bereits eine Kopie erhalten hat. Um das zu ermöglichen, wie auch um Routing-Loops zu vermeiden, verwendet AODV Folgenummern, die in allen Paketen enthalten sind.

Wenn „Route-request“-Pakete weitergeleitet werden fügt der weiterleitende Knoten den Knoten, von dem er das Paket erhalten hatte, in seine Routingtabelle ein. Diese Information wird genutzt, um den umgekehrten Pfad zu generieren, wenn das „Route-reply“-Paket übermittelt wird. Wegen dieses Mechanismus kann AODV nur bidirektionale Links benutzen. Wenn das „Route-reply“-Paket dem Pfad zurück zur Quelle, die den „Route discovery“-Prozeß initiiert hatte, folgt, fügt jeder Knoten den Pfad in seine Tabelle ein.

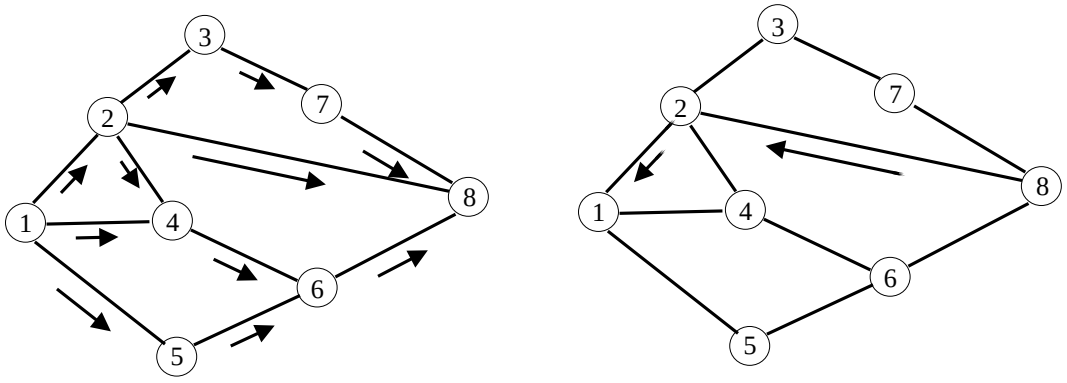

Abbildung 5: „Route query“ und „Route reply“ beim „Route discovery“-Prozess von Knoten 1 nach Knoten 8

3.3.4 Associativity-Based Routing

Das Associativity-Based Routingprotokoll (ABR) beruht auf einem völlig anderen Ansatz. Es führt einen *Stabilitätsgrad* als einen neuen Routingparameter ein und vollzieht Routing gemäß des Stabilitätsgrads der Knoten. Der Stabilitätsgrad eines Knotens in Bezug auf einen

anderen ist definiert als die Stabilität seiner Verbindung über Raum und Zeit zum anderen. [RoTo00] Das zugrundeliegende Ziel von ABR ist es, Pfade zur Verfügung zu stellen, die verlässlicher sind weil dauerhafter.

Jeder Knoten erzeugt periodisch „Beacons“, um über seine Existenz zu informieren. Nachbarn erhöhen ihren Stabilitätsgrad des sendenden Knotens mit jedem Beacon, den sie erhalten.

„Route discovery“ wird duch einen „Query“ und „Reply“ Vorgang initiiert. Knoten, die den „Query“ erhalten und nicht Ziel sind, hängen ihre Adressen und ihre Stabilitätsgrade mit ihren Nachbarn an das „Query“-Paket. Ein folgender Knoten löscht die Stabilitätsgrade des Vorgängers mit seinen Nachbarn und erhält ausschließlich den Eintrag, der ihn und seinen Vorgänger betrifft. So enthält jedes Paket, das am Ziel ankommt, die Stabilitätsgrade der Knoten entlang des Pfades zum Ziel. Das Ziel ist dann in der Lage, den besten Pfad auszuwählen. Im Anschluß sendet es ein „Reply“-Paket zurück zur Quelle, entlang dieses Pfades. Knoten, die das „Reply“-Paket weiterleiten, markieren den entsprechenden Pfad als gültig. [RoTo00]

Wenn sich die Netzwerktopologie ändert, wird ein Pfad entweder durch die Initiierung eines neuen „Route discovery“-process oder durch ein teilweises Update des alten Pfades wiederhergestellt. Wenn kaputte Pfade entdeckt werden wird eine „Route delete“-Nachricht durch das Netzwerk geleitet.

4 Vergleich

Selbst wenn Simulationen durchgeführt wurden, um die verschiedenen Routingprotokolle zu testen, haben diese selten dieselben Charakteristika unter denselben Bedingungen untersucht. Material, das den Vergleich verschiedener Protokolle möglich macht, ist kaum erhältlich. Generelle Aspekte können jedoch verglichen werden [RoTo00].

4.1 Die Table-driven-Routingprotokolle

DSDV, CGSR und WRP nutzen alle einen „shortest path“-Routingparameter. DSDV ist eher ineffizient, da es periodische Routingtabellen-Updates erfordert, auch wenn sich die Netzwerktopologie nicht ändert. Dies setzt der Größe des Netzwerks eine Grenze. [RoTo00]

CGSR basiert auf DSDV. CGSR benötigt eine Clusterheadtabelle zusätzlich zur Routingtabelle. Auf der anderen Seite benutzt es, im Gegensatz zu DSDV und WRP ein hierarchisches Routingschema, das Routing effizienter macht. Der Nachteil des hierarchischen Routing ist, dass der Prozeß der Wahl des Clusterheads Ressourcen verbraucht. Außerdem sind die Clusterheads kritische Knoten, weil ihr Ausfall das Netzwerk in größerem Maße beeinträchtigt als der Ausfall eines beliebigen Knoten, wenn eine flache Routingphilosophie zugrundegelegt wird.

WRP unterscheidet sich von DSDV und CGSR. Jeder Knoten muß vier verschiedene Tabellen pflegen, was besonders in einem größeren Netzwerk hohe Speicherkapazitäten erfordert. Außerdem erfordert WRP im Gegensatz zu den anderen beiden das Senden von *Hello*-Paketen, was Bandbreite benötigt. Daneben verhindert es, dass ein Knoten in den „Sleep“-Modus übergeht.

Während WRP, DSDV und CGSR im allgemeinen frei von Loops sind, hat WRP einen Vorteil gegenüber den anderen beiden: Es vermeidet auch temporäre Routing-Loops indem es die Konsistenz seiner Routinginformation mit seinen Nachbarn überprüft.

Fällt ein Link aus, so braucht WRP weniger Schritte um zu konvergieren als DSDV, denn es informiert nur seine Nachbarn über den kaputten Link. Im Gegensatz dazu braucht CGSR mehr Schritte, da es darüber hinaus die Neuwahl der Clusterheads durchführen muß.

Die Anzahl der Nachrichten, die benötigt wird, wenn ein Link ausfällt ist mehr oder weniger dieselbe bei allen drei Protokollen, da alle auf „distance vector shortest-path routing“ beruhen.

Keines der drei Protokolle unterstützt Multicast, wenn es auch ein Protokoll gibt, das Multicast unterstützt und auf CGSR läuft. [RoTo00]

4.2 Die On-demand-Routingprotokolle

AODV benutzt einen „Route discovery“-Prozeß ähnlich dem von DSR. Im Gegensatz zu AODV, das nur die Zieladresse enthält, enthält unter DSR jedes Paket sämtliche Routinginformation in seinem Header, was einen größeren Overhead bedeutet. Dies gilt ebenso für den „Reply“-Prozeß.

Außerdem benötigt DSR mehr Speicherplatz, da es die Speicherung ganzer Pfade vorsieht, während AODV sich auf Information zum nächsten Hop beschränkt. Da DSR auf Source-Routing basiert, kann es auf eine wesentlich größere Menge an Routinginformation zurückgreifen, als AODV. Unter DSR erhält die Quelle sämtliche Information über die intermediären Knoten, ebenso wie diese sämtliche Information mithören. Im Gegensatz dazu kann AODV nur eine begrenzte Menge an Routinginformation zusammentragen, da es kein Source-Routing benutzt und keine Information über Mithören gewinnt. Dies ist der Grund dafür, dass AODV öfter auf „Route discovery“ Mechanismen zurückgreifen muß, was einen großen Overhead bedeutet. [DaPR00]

Außerdem ist AODV das einzige Protokoll der diskutierten, das eine Option für Multicast bereitstellt. DSR hat im Vergleich mit AODV den Vorteil, dass es unidirektionale Links benutzen kann. [RoTo00]

Verglichen mit anderen On-demand-Protokollen, spart DSR Bandbreite und reduziert den Energieverbrauch indem es keine periodischen Routingtabellen-Updates erfordert, wenn sich die Topologie nicht ändert. Außerdem sieht es die Speicherung von mehr als einem Pfad zu einem Ziel vor. Das bedeutet im Allgemeinen eine schnellere Konvergenz im Falle unterbrochener Links da es nicht nötig ist, neue „Route discovery“-Prozesse einzuleiten, wenn Alternativen gespeichert sind.

Im Gegensatz zu DSR, das keine Mechanismen besitzt, um überholte Pfade zu eliminieren, wählt AODV mit Hilfe von Folgenummern immer den aktuelleren Pfad. [DaPR00] Während unter AODV ein „Error“-Paket sämtliche Knoten, die einen unterbrochenen Link benutzen, erreicht, verfolgt ein „Error“-Paket unter DSR ausschließlich das Paket zurück, dass den ausgefallenen Link entdeckt. [DaPR00] Im Gegensatz zu AODV benötigt DSR keinen Timer.

Der Vergleich von DSR und AODV mittels Simulation zeigt, dass für anwendungsorientierte Parameter wie Verzögerung und Durchsatz, DSR in weniger stressvollen Situationen besser abschneidet, während AODV in stressvolleren (d.h. zum Beispiel größeres Datenvolumen oder höhere Mobilität) weit bessere Ergebnisse liefert. [DaPR00]

Aufgrund seines hohen Overhead im Routingprozeß wird DSR nicht als angemessen für Netzwerke mit einer größeren Anzahl von Knoten betrachten.

Das Protokoll, das Netzwerke bestehend aus einer Vielzahl von Knoten am besten bewältigt, ist TORA [RoTo00]. TORA erlaubt, wie von den besprochenen Routingprotokollen sonst nur noch DSR, die Speicherung mehrerer Pfade zu einem Ziel. Selbst wenn TORA selbst keine

Möglichkeit zu Multicast bietet, besitzt es die Option, ein anderes Protokoll „aufzusetzen“ (LAN), das Multicast erlaubt.

Ein Nachteil von TORA ist, dass es von einer externen Zeitquelle wie GPS abhängt. Dies macht das Protokoll verletzlich in Bezug auf Situationen in denen das Zeitsystem einzelnen Knoten oder sogar dem gesamten Netzwerk nicht mehr zur Verfügung steht.

Auch funktioniert die Wiederherstellung von Pfaden unter TORA im Allgemeinen nicht so schnell, wie unter anderen Algorithmen. [RoTo00]

ABR hat den Vorteil gegenüber anderen Protokollen, dass es die Dauerhaftigkeit von Pfaden berücksichtigt. Während es bei der Anzahl der Hops Opfer bringt, glänzt es mit höherem Durchsatz wegen weniger Wiederherstellung von Pfaden.

Ein kritischer Aspekt von ABR ist das Erfordernis von ständigem Übermitteln von „Beacons“, was zusätzlichen Energieverbrauch bedeutet, auch wenn es Simulationen gibt, die zeigen, dass das nicht bedeutend ist. [RoTo00]

ABR erlaubt intermediären Knoten im Gegensatz zu DSR und AODV zu versuchen, unterbrochene Pfade selbst wiederherzustellen. Dies vermeidet die erneute Initiation von „Route discovery“-Prozessen durch die Quelle, wenn es eine Möglichkeit gibt, den Pfad auf lokaler Ebene wiederherzustellen. Dies führt zu einer geringeren Konvergenzzeit. Wenn jedoch intermediäre Knoten versuchen, einen Pfad wiederherzustellen und ihnen das nicht gelingt, so dass danach die Quelle doch einen neuen „Route discovery“-Prozess initiieren muß, war das ganze im Nachhinein kostspieliger in Bezug auf Konvergenzzeit, als wenn keine Möglichkeit der Wiederherstellung von Pfaden durch intermediäre Knoten vorgesehen war.

Wie die Table-driven-Protokolle sind all die genannten On-demand-Protokolle frei von Loops. Alle benutzen flache Netzwerkorganisation. Während alle „Shortest-path-Routing“ verwenden, berücksichtigt AODV zusätzlich die Aktualität der Pfade und ABR deren Stabilität.

Der Vorteil von Table-driven- im Vergleich mit On-demand-Protokollen ist, dass sie Routinginformation bereits besitzen, wenn sie sie brauchen und es nicht nötig ist, einen „Route discovery“-Prozeß zu initiieren bevor sie senden können. Der große Nachteil ist, dass die ständige Pflege dieser Routinginformation in Bezug auf Bandbreite und Energieverbrauch mit hohen Kosten verbunden ist. Da sowohl Bandbreite als auch Energie in Ad-Hoc-Netzwerken knappe Ressourcen darstellen, ist dies eine ernstzunehmende Einschränkung. [RoTo00]

5 Schlußbemerkung

Während sowohl Table-driven- als auch On-demand-Protokolle ihre Vorteile haben, genügen On-demand-Protokolle den Anforderungen von Ad-Hoc-Netzwerken im Allgemeinen besser. Da sie den durchschnittlichen Netzwerkverkehr gering halten, reduzieren sie den Verbrauch an Bandbreite und Energie. Wenn dann tatsächlich ein Pfad gefunden werden muß, ist diese Variante kostspieliger. Diese Philosophie, Bandbreite und Energie nur dann zu verbrauchen, wenn ein Pfad gebraucht wird, kommt Ad-Hoc-Netzwerken zu Gute.

Im Allgemeinen ist die Entwicklung von Ad-Hoc-Netzwerktechnologie noch in ihren Anfängen. Dies trifft ebenso in Bezug auf Routing zu. Keines der existierenden Protokolle ermöglicht generell optimales Routing, die meisten bewältigen den Mobilitätsaspekt von Ad-Hoc-Netzwerken nur unter bestimmten, begrenzten Bedingungen.

Andere Aspekte sind noch gar nicht einbezogen oder werden nur ungenügend bewältigt. So berücksichtigt zum Beispiel keines der besprochenen Routingprotokolle die begrenzten Batterieressourcen bei der Wahl eines Pfads [ChTa00]. Außerdem sind Sicherheitsmechanismen

in Ad-Hoc-Routingprotokollen immer noch nicht vorgesehen oder nachträglich unzureichend angepasst. [Kärp00]

Es besteht ebenfalls ein großer Bedarf an Simulationen um die existierenden Protokolle vergleichen zu können und ihre Probleme und damit ihr Potential zu Verbesserungen auszumachen. „The field of ad-hoc mobile networks is rapidly growing and changing, and while there are still many challenges that need to be met, it is likely that such networks will see wide-spread use within the next few years.“ [RoTo00]

Literatur

[ChTa00] Jae-Hwan Chang und Leandros Tassiulas. Energy Conserving Routing in Wireless Ad-hoc Networks. In *Proceedings of IEEE INFOCOM00*, 2000.

[DaPR00] Samir R. Das, Charles E. Perkins und Elizabeth M. Royer. Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. In *Proceedings of IEEE INFOCOM00*, 2000.

[Kärp00] Vesa Kärpijoki. Signalling and Routing Security in Mobile and Ad-hoc Networks. http://www.hut.fi/~vkarpijo/iwork00/, 2000.

[Lesi98] B.Cameron Lesiuk. Routing in Ad Hoc Networks of Mobile Hosts. http://phantom.me.uvic.ca/clesiuk/thesis/reports/adhoc/adhoc.html, 1998.

[LiHa00] Ben Liang und Zygmunt J. Haas. Virtual Backbone Generation and Maintenance in Ad Hoc Network Mobility Management. In *Proceedings of IEEE INFOCOM00*, 2000.

[RoTo00] Elizabeth M. Royer und C.-K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Network. http://www.ee.surrey.ac.uk/Personal/G.Aggelou/PAPERS/Adhoc_Review.ps.gz, 2000.

Abbildungsverzeichnis