ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Active Network Service mit Netlink

Semesterarbeit von Julio Pérez
Oktober 2003 – Februar 2004

Betreuer: Matthias Bossardt
Professor: Bernhard Plattner

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Zusammenfassung

Aktive Knoten zeichnen sich dadurch aus, dass sie zusätzlich zum routen von Paketen flexible, sogenannte aktive Services anbieten. In dieser Arbeit wurde solch ein aktiver Service implementiert, der auf einem Click Router unter Verwendung der Chameleon Deployment-Architektur installiert und ausgeführt wird. Die Aufgabe des Services besteht darin, über mehrere erhaltene Pakete hinweg Informationen zu sammeln und zusammenzufassen, um die so aggregierten Daten an einen bestimmten Zielknoten zu schicken. In der Literatur wird dafür der Begriff Information-Fusion verwendet. Der Service setzt sich sowohl aus Komponenten zusammen, die in der Click Execution Environment (Click EE) ausgeführt werden, wie auch aus solchen, die in der Chameleon Java EE ablaufen.

Für die Kommunikation zwischen den EEs wurde Netlink verwendet. Da Chameleon bei Beginn dieser Arbeit für die Inter-EE Kommunikation "nur" das Proc-Filesystem anbot, wurden Adapter-Module implementiert, welche die Verwendung von Netlink für die Inter-EE Kommunikation erlauben.

Inhaltsverzeichnis

Liste der Abbildungen und Tabellen

Kapitel 1

Einleitung

In diesem Kapitel wird kurz auf den für diese Arbeit verwendeten Router, der sogenannte Click Router, eingegangen. Der darauffolgende Abschnitt beschreibt die Chameleon Deployment-Architektur, bevor zum Schluss der Aufbau dieses Dokumentes umrissen wird.

Click [4] ist eine Software-Architektur, die dazu verwendet wird, flexible und konfigurierbare Router zu bilden. Ein Click Router ist aus mehreren paket-abarbeitenden Software-Modulen zusammengesetzt, sogenannte Elemente. Individuelle Elemente implementieren einfache Routerfunktionen wie Kommunikation mit Netzwerkschnittstellen, Paketklassifizierung, Queueing etc. Weiter können Elemente beliebig viele In- und / oder Outports haben, welche es erlauben, Elemente miteinander zu verbinden. Jede Verbindung führt dabei vom Outport eines Elementes zum Inport eines anderen. Daraus resultiert schlussendlich ein gerichteter Graph (siehe Abbildung 1.1), der die eigentliche Routerkonfiguration darstellt. Die so gebildete Konfiguration gibt demnach an, von welchen Elementen und in welcher Reihenfolge Pakete abgearbeitet werden.

Click unterstützt drei Arten von Verbindungen zwischen Elementen. Bei einer Push-Verbindung initiiert das Quell-Element den Pakettransfer, bei einer Pull-Verbindung entsprechend das Ziel-Element (d.h. es fordert das Quell-Element auf, ein Paket zu liefern, oder einen Null-Pointer wenn keines anliegt). Ein Agnostic-Port kann sowohl mit einem Pull- wie auch mit einem Push-Port verbunden sein.

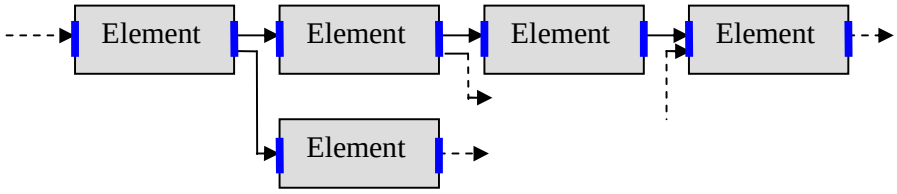

Abbildung 1.1: Veranschaulichung einer Click-Konfiguration

Damit ein aktiver Knoten Services anbieten kann, muss er mit einer Service Deployment-Architektur ausgestattet sein. Diese erlaubt bei Ankunft eines Service-Requests, diejenigen Module zu indentifizieren und zu installieren, welche notwendig sind, um den verlangten Service realisieren zu können. Dieser Arbeit lag als Deployment-Architektur Chameleon zugrunde, welche am Institut für Technische Informatik und Kommunikationsnetze (TIK) der ETH Zürich entwickelt wurde. Im wesentlichen besteht Chameleon aus drei Hauptkomponenten: Aus zwei Execution Environments (EEs), also Ausführungsumgebungen, nämlich der sogenannten Click EE (CEE) und der Chameleon Java EE (CJEE), und aus der Service Creation Engine (SCE).

Die Komponenten eines Services laufen in den EEs ab. Dabei basiert die CEE, wie der Name schon verrät, auf dem Click Router. Als Programmiersprache dient C++, jedes Click-Element setzt sich also aus einer C++-Klasse und einer zugehörigen Header-Datei zusammen. Die wohl wichtigste Eigenschaft der CEE ist, dass der Code im Kernelspace ausgeführt wird, weshalb sich diese Ausführungsumgebung besonders für zeitkritische Anwendungen eignet. Im Gegensatz dazu basiert die CJEE auf der Java Virtual Machine, der Code wird im Userspace ausgeführt. CJEE-Komponenten haben ebenfalls In- bzw. Outports, welche mit den Ports von CJEE- wie auch von CEE-Komponten verbunden werden können. Damit aber über EE-Grenzen hinaus kommuniziert werden kann, sind spezielle Adapterelemente notwendig, die diese Art von Kommunikation regeln. Im nächsten Kapitel wird detaillierter darauf eingegangen.

Chameleon’s dritte Komponente, die Service Creation Engine, hat die Aufgabe, einen knotenunabhängigen Service-Request (in XML-Format) auf die Architektur desjenigen Knotens abzubilden, auf dem die SCE läuft. Sie indentifiziert dabei jene Komponenten die erforderlich sind, um den verlangten Service bereitzustellen, und lädt bzw. installiert sie in die jeweiligen EEs. Ausserdem ist die SCE für die Konfiguration beider EEs verantwortlich. Ein Beispiel für einen Service-Request befindet sich im Anhang B (Abbildung B.5).

Der Rest dieses Dokumentes ist folgendermassen aufgebaut:
In Kapitel zwei wird die Netlink Inter-EE Kommmunikation behandelt. Kapitel drei befasst sich mit dem eigentlichen Service. Dazu gehört eine kurze Beschreibung des Service-Typs, bevor detailliert auf den Aufbau und die verwendeten Komponenten eingegangen wird. Im letzten Kapitel befinden sich eine kurze Zusammenfassung dieser Arbeit und der gesammelten Erfahrungen. Ausserdem findet man dort Vorschläge für Verbesserungen bzw. Anregungen für zukünftige Arbeiten.

Kapitel 1: Einleitung

Kapitel 2

Inter-Execution-Environment Kommunikation mit Netlink

Dieses Kapitel befasst sich kurz damit, welche Möglichkeiten es für die Kommunikation zwischen User- und Kernelspace in Linux gibt. Ausserdem wird erläutert, wie Chameleon erweitert wurde, um für die Inter-EE Kommunikation Netlink verwenden zu können.

2.1 Inter-EE Kommunikationsmöglichkeiten

Für die Kommunkation zwischen Kernel-Modulen und Userspace-Prozessen bietet Linux unter anderem das Proc-Filesystem an. Üblicherweise befindet sich das Proc-Filesystem unter /proc bzgl. des Root-Verzeichnisses und kann etwas poetisch ausgedrückt als Fenster zur Seele des Systems bezeichnet werden. Darin findet man praktisch alles was man über den Status und die Konfiguration eines Rechners wissen will, wie z.B. Statusinformationen über Netzwerk-Protokolle, welche Kernel-Module im Moment geladen sind etc. Es ist jedoch auch möglich, dass Kernel-Module und Userspace-Prozesse über dieses virtuelle Filesystem relativ einfach Informationen austauschen bzw. miteinander kommunizieren.

Obwohl das Proc-Filesystem recht angenehm zu benutzen ist, besteht doch manchmal das Bedürfnis, eine andere, "direktere" Art von Kommunikation zu benützen. Dazu bietet sich Linux Netlink an, das eine bidirektionale Kommunikation zwischen Kernel- und Userspace anbietet. Es besteht aus einer socketbasierten Standardschnittstelle für User-Prozesse und einem internen Kernel-API für Kernel-Module. Für genauere Angaben zu Netlink sei auf [5] verwiesen.

Damit die angesprochenen Kommunikationsmöglichkeiten zwischen Kernel- und Userspace für den Inter-EE Informationsaustausch auf einem Chameleon-Knoten

gebraucht werden können, sind Adapter-Module notwendig. Diese sorgen dafür, dass Daten zu anderen EEs gesendet bzw. von anderen EEs empfangen werden.

Für die Inter-EE Kommunikation mit Hilfe des Proc-Filesystems sind zwei solche Adapter notwendig, die beide in der CJEE ausgeführt werden: Wenn der Konfigurator der CJEE eine Verbindung von einer anderen EE zur CJEE feststellt, fügt er ein sogenanntes FromFile-Element ein. Dieses liest Daten aus einer bestimmten Proc-Filesystem-Datei, die vorher von einem CEE-Element beschrieben wurde. Für die andere Richtung, d.h. von der CJEE zur CEE, wird ebenfalls vom CJEE-Konfigurator das ToFile-Element eingefügt, welches von einem Userspace-Element erhaltene Daten in eine Proc-Filesystem-Datei schreibt, auf die nachfolgende Kernel-Module lesend zugreifen können (siehe Abbildung 2.1).

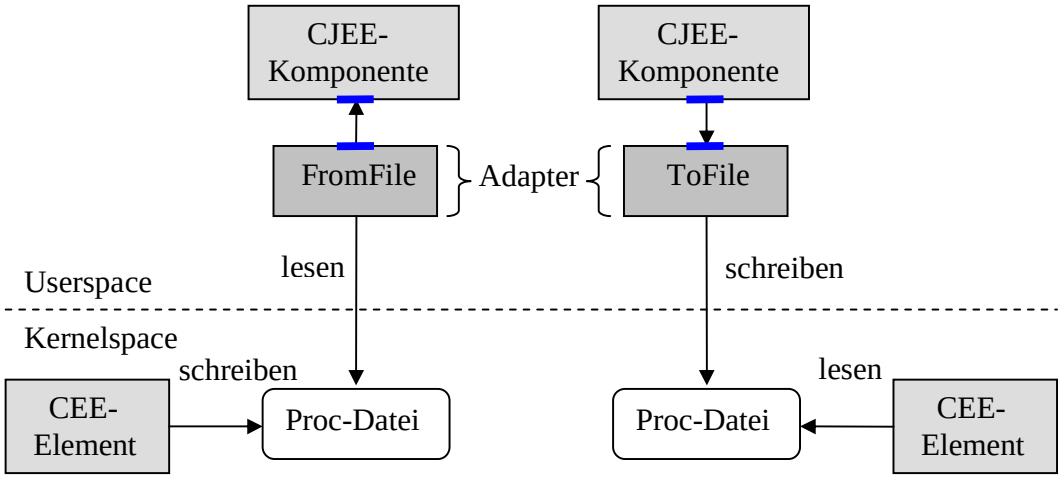

Abbildung 2.1: Inter-EE Kommunikation mit Proc-Filesystem

Wenn man für die Inter-EE Kommunikation Netlink verwenden möchte, sind verständlicherweise andere Adapter-Module notwendig. Entsprechend müssen die EE-Konfiguratoren so abgeändert werden, dass diese Module und nicht diejenigen für die Kommunikation über das Proc-Filesystem benutzt werden.

Prinzipiell sind drei Adapter nötig: Auf der Kernelspace-Seite sind zwei Module von Belang, nämlich die To- bzw. FromNetlink genannten Elemente. Auf der Userspace-Seite trägt der JNetlink benannte Adapter zur Inter-EE Kommunikations-ermöglichung bei. Es sind also neu sowohl CEE- wie auch CJEE-Komponenten nötig. Welche Aufgaben die angesprochenen Adapter-Module haben und welche

sonstige Änderungen an der SCE vorgenommen werden mussten, wird in den nächsten zwei Unterkapiteln behandelt.

2.2 Netlink-Support für Click EE

Anders als bei der Inter-EE Kommunikation über das Proc-Filesystem, wo die CEE nicht speziell konfiguriert werden muss, da Click-Elemente sowieso fähig sind über dieses Filesystem mit anderen Elementen Daten auszutauschen, muss bei Verwendung von Netlink der CEE-Konfigurator angepasst werden. Es muss dafür gesorgt sein, dass bei einer Verbindung, welche von einer Click- zu einer Chameleon Java-Komponente führt, das ToNetlink-Element zwischengeschaltet wird. Dieses Modul besitzt die Aufgabe, die von einer Click-Komponente erhaltenen Pakete über einen zuvor erzeugten Netlink-Socket Richtung Userspace zu schicken. Abbildung 2.2 veranschaulicht diesen Vorgang. Über Parameter kann zusätzlich spezifiziert werden, an wen genau die Pakete überreicht werden sollen: Wird eine Prozess-ID angegeben, folgt ein Unicast an den entsprechenden Prozess. Wird hingegen keine Prozess-ID spezifiziert, oder sie ist gleich Null gesetzt, folgt ein Multicast an die ebenfalls durch Parameter angegebene Gruppe.

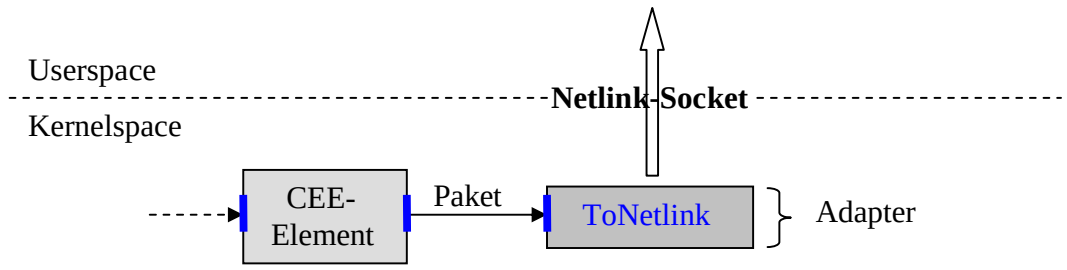

Abbildung 2.2: Adapter-Modul ToNetlink

Bei einer entgegen gerichteten Verbindung, d.h. von einer Chameleon Java- zu einer Click-Komponente, kommt das FromNetlink-Element zum Einsatz. Es dient dazu, die vom Userspace über einen ebenfalls zuvor erzeugten Netlink-Socket erhaltenen Daten zu einem Paket zusammenzufassen, um dieses anschliessend an die nachfolgende Click-Komponente weiterzureichen (siehe Abbildung 2.3). Leider ist es bis zur Beendigung dieser Arbeit nicht gelungen, FromNetlink fertigzustellen. Es kann zwar kompiliert und geladen werden, bekommt jedoch nie Pakete überreicht. Es scheint

auch recht schwierig zu sein, Informationssmaterial zum Netlink Kernel-API zu finden.

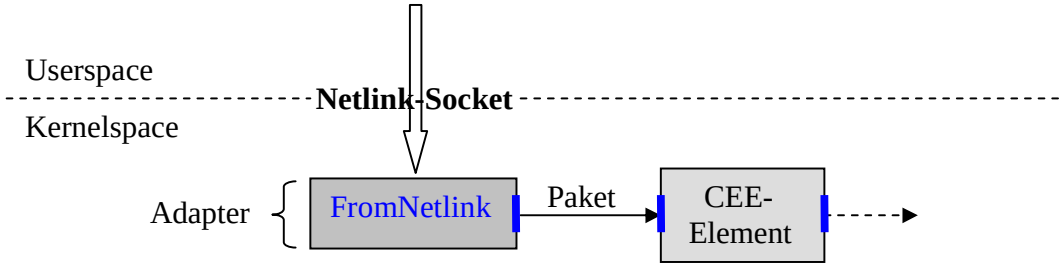

Abbildung 2.3: Adapter-Modul FromNetlink

2.3 Netlink-Support für Chameleon Java EE

Mit den beiden Click-Adaptern wäre eine Inter-EE Kommunikation noch nicht möglich, da aus einer in der CJEE ausgeführten Java-Applikation nicht auf ein Netlink-Socket zugegriffen werden kann. Hierfür wurde der JNetlink-Adapter eingeführt, welcher aus einem vorher erzeugten Netlink-Socket jene Daten ausliest, die vom ToNetlink-Adapter verschickt wurden. Die so erhaltenen Daten werden dann über das Java Native Interface[1] (JNI) an die Java-Applikation übergeben. Für die Kommunikation in die andere Richtung könnte die Java-Komponente, wieder unter Gebrauch des JNI, Daten an JNetlink überreichen, welcher dann mittels eines zweiten Netlink-Sockets die Daten an das FromNetlink-Modul verschicken würde (wie schon erwähnt klappt diese Kommunikation leider nicht und wurde deshalb nicht in Abbildung 2.4 berücksichtigt). Es sei nochmals explizit darauf hingewiesen, dass sich bei Verwendung von Netlink die CJEE-Konfiguration wesentlich vereinfacht, da JNetlink komplett unabhängig von der CJEE ist und der Konfigurator keine Java-Adapter einfügen muss. Den Preis den man dafür zahlen muss ist, dass sich andererseits die Click EE-Konfiguration etwas erschwert, verglichen mit der Konfiguration bei Verwendung des Proc-Filesystems.

Mit der Einführung von neuen Adapter-Komponenten und der Modifizierung der EE-Konfiguratoren ist die Arbeit allerdings noch nicht ganz getan. Es muss noch eine kleine, zusätzliche Änderung vorgenommen werden. Die SCE konsultiert die sogenannte Node Description, um zu überprüfen, ob ein verlangter Service auf dem Knoten, auf dem die SCE läuft, angeboten werden kann. Die Node Description

[1] Ein gutes Tutorial findet man unter [6]

enthält Informationen über das Betriebssystem des Knotens, welche EEs angeboten werden, Anzahl und Typen der EE-Ports und in diesem Zusammenhang am wichtigsten, welche Art von Verbindungen zwischen den EEs erlaubt sind. Gemäss ursprünglicher Node Description konnten für die Verbindung von der CEE zur CJEE nur Procfs_Outports[2] mit Push-Inports verbunden werden. Da aber neu auch Netlink für die Inter-EE Kommunikation verwendet werden können soll, mussten zusätzliche Verbindungsmöglichkeiten angegeben werden, konkret Agnostic-Outports zu Push-Inports (siehe Anhang B, Abbildung B.1). Analog müssten für die Verbindung von der CJEE zur CEE zusätzliche Möglichkeiten eingetragen werden, wenn man FromNetlink integrieren möchte.

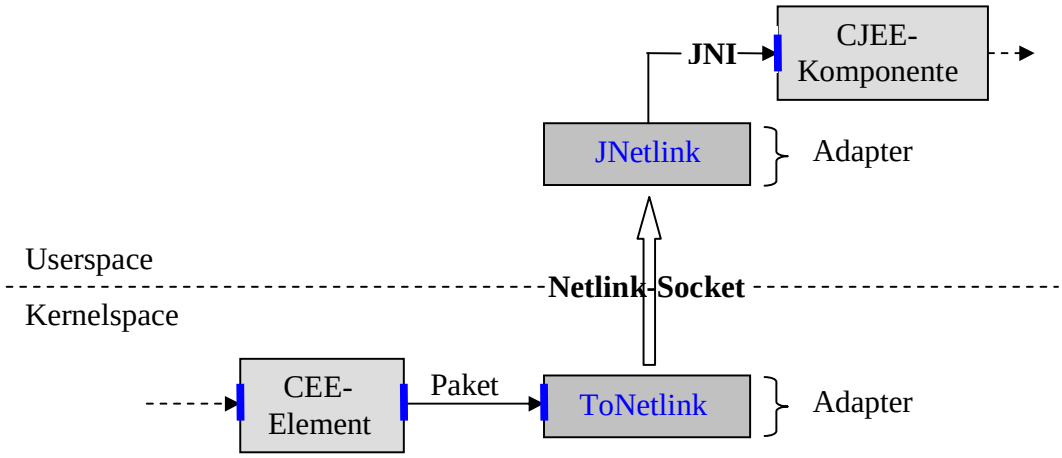

Abbildung 2.4: Inter-EE Kommunikation mit Netlink und JNI

[2] Dieser Port-Typ wurde für die Inter-EE Kommunikation über das Proc-Filesystem eingeführt, existiert aber nicht in Click.

Kapitel 2: Inter-Execution-Environment Kommunikation mit Netlink

Kapitel 3

Information-Fusion Service

In diesem Teil der Dokumentation wird zu Beginn kurz auf den gewählten Servicetyp eingegangen. Anschliessend folgt eine detaillierte Beschreibung des implementierten Services und aller anderweitig gebrauchten (Hilfs-)Komponenten.

3.1 Information-Fusion

Es gibt grundsätzlich eine Vielzahl von Services die sich für einen aktiven Knoten anbieten. Mögliche Bespiele sind unter anderem Firewalls, Web-Caching, spezielle Multicast-Protokolle, Transcoding etc. Die aufgezählten Services haben miteinander gemein, dass sie immer wieder in der Literatur anzutreffen sind und deshalb für diese Arbeit weniger von Interesse waren. Sattdessen fiel die Wahl auf einen anderen Servicetyp, der in der heutigen Zeit, wo es immer mehr mit dem kontinuierlich ansteigenden Netzwerkverkehr zu kämpfen gilt, von wichtiger Bedeutung ist: Information-Fusion.

Kurz und prägnant ausgedrückt, beschäftigt sich Information-Fusion damit, von verschiedenen Quellen erhaltene Daten zu filtern und so gut wie möglich zusammenzufassen. Das Ziel ist es, alle irrelevanten und veralteten Informationen auszusortieren, um so das Datenvolumen zu reduzieren, wobei man natürlich darauf achten muss, dass nicht auch relevante Daten entfernt werden. Das aggregierte Resultat wird dann an einen zentralen Knoten geschickt.

Dabei kann Information-Fusion auf Daten jeglicher Art angewendet werden. Die ursprüngliche Idee war, einen Information-Fusion Service für Finanz- bzw. Börsendaten zu implementieren. Dabei würde man Werte von Aktien, die an verschiedenen Börsen gehandelt werden, miteinander vergleichen, um dann z.B. zu bestimmen, wo die Aktie am höchsten gehandelt wird. Das Problem dabei ist, dass man erstens schlecht gratis an Finanzdaten herankommt, und zweitens, dass es sich dabei um kritische Daten handelt. Letzteres ist deswegen ein Problem, weil wichtige

Daten jeder Zeit im Netz verloren gehen können, so dass das Resultat eventuell sehr stark verfälscht wird. Deshalb fiel die Entscheidung schlussendlich auf einen Monitoring Information-Fusion Service, d.h. es werden Daten zusammengefasst, die Informationen über Knoten und Verbindungen in einem Netzwerk enthalten.

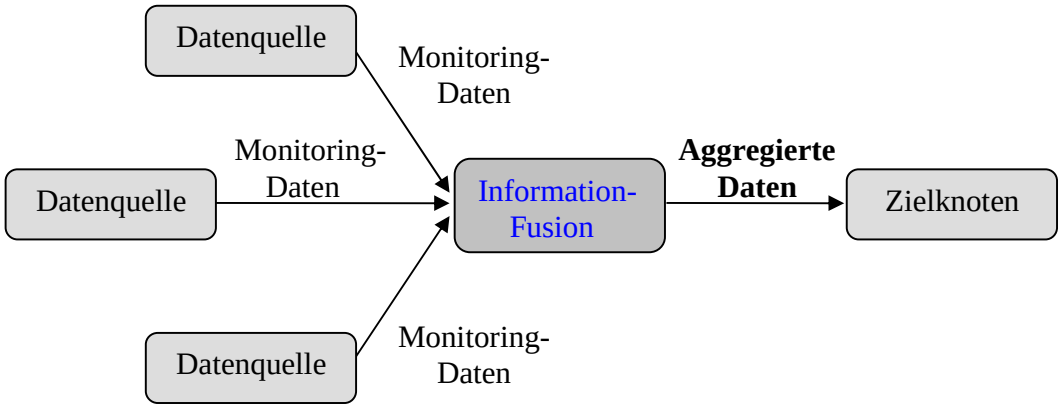

Abbildung 3.1: Information-Fusion

3.2 Aufbau und Struktur des Services

Der Aufbau des ganzen Services ist in Abbildung 3.2 dargestellt. Es werden prinzipiell sechs Komponenten verwendet: 4 Kernel-Module, allesamt Click-Elemente, wovon einer ein Adapter-Modul ist (für die Inter-EE Kommunikation), und zwei Userspace-Komponenten, nämlich wiederum ein Adapter und eine Chameleon Java EE-Komponente. Welche Aufgaben die jeweiligen Komponenten haben, wird in den nächsten Unterkapiteln behandelt.

Wie schon weiter oben erwähnt, ist es nicht gelungen, das FromNetlink-Adapter-Modul funktionstauglich zu machen. Trotzdem wurde überlegt, welche Komponenten zusätzlich nötig oder von Vorteil wären, wenn man für die Kommunikation von der CJEE zur CEE Netlink verwenden könnte. Das UDPIPEncap-Element[3] würde ein von FromNetlink erhaltenes Paket um einen UDP- und einen IP-Header erweitern. EtherEncap würde dann dem resultierenden Paket einen Ethernet-Header hinzufügen, bevor ToDevice das Ganze an eine Linux-Netzwerkschnittstelle verschicken würde.

[3] Die Dokumenation zu den Click-Elementen UDPIPEncap, EtherEncap und ToDevice findet man unter http://www.pdos.lcs.mit.edu/click/doc

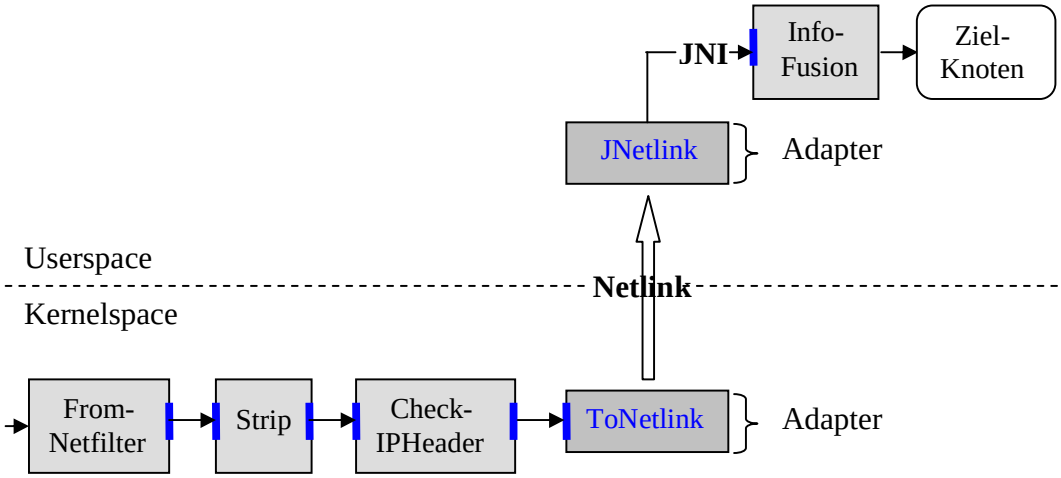

Abbildung 3.2: Service-Aufbau

3.3 Service-Komponenten

Nun da bekannt ist, welche Aufgaben der Service hat und wie er zusammengesetzt ist, kann man näher auf die einzelnen Komponenten eingehen. Diese können prinzipiell in zwei Klassen unterteilt werden, nämlich in Hilfskomponenten, die nichts mit der eigentlichen Information-Fusion zu tun haben, und in solche, die genau dafür implementiert wurden.

3.3.1 Verwendete Hilfskomponenten

Bevor sich die eigentliche Information-Fusion abspielt, müssen ein paar Komponenten den Inhalt eines Paketes bearbeiten bzw. auf gewisse Eigenschaften überprüfen.

Das FromNetfilter-Click-Element hat die Aufgabe, Pakete von Netfilter, dem Linux Paketfilter, entgegenzunehmen und weiterzureichen. Dafür registriert es eine Callback-Funktion, welche aufgerufen wird, wenn gewisse Netfilter-Regeln auf Pakete zutreffen. Dabei sind die Regeln, nach denen die Pakete gefiltert werden sollen, entweder im Service-Request oder in der XML-Beschreibung des Services angegeben (siehe Anhang B, Abbildung B.2).

FromNetfilter fügt den Paketen einen Ethernet-Header hinzu, für den implementierten Service sind diese zusätzlichen Daten jedoch irrelevant, weshalb das Strip-Element die ersten 14 Bytes des erhaltenen Paketes wieder entfernt.

Zusätzlich zu den Überprüfungen, die Netfilter durchführt, versichert sich CheckIPHeader, dass die Header-Länge, die gesamte Länge und die Prüfsumme des Paketes korrekt sind.

Nachdem das Paket von den erwähnten Click-Elementen bearbeitet wurde, müssen die resultierenden Daten Richtung Userspace geschickt werden. Hier kommen die in Kapitel zwei besprochenen Adapter-Module zum Einsatz. Konkret nimmt ToNetlink Pakete von CheckIPHeader entgegen und schickt die darin enthaltenen Daten über einen Netlink-Socket an JNetlink. Dieser reicht dann via Java Native Interface die Daten an die InfoFusion-Komponente, die im nächsten Abschnitt besprochen wird, weiter.

Es sei an dieser Stelle darauf hingewiesen, dass FromNetfilter und die Adapter-Module implizit durch die Service Creation Engine eingefügt werden, weshalb sie in der Service-Beschreibung nicht angegeben werden sollten.

3.3.2 Information-Fusion Komponente

Die bisher behandelten Komponenten haben eigentlich nichts mit der grundsätzlichen Aufgabe des Services, sprich mit Information-Fusion, zu tun. Dafür ist einzig und allein die InfoFusion genannte CJEE-Komponente, welche in diesem Abschnitt beschrieben werden soll, verantwortlich.

3.3.2.1 Datenformat der Pakete

Bevor näher auf den Algorithmus eingegangen wird, sollte besprochen werden, welches Datenformat vorausgesetzt wird. Wie schon in Abschnitt 3.1 erwähnt, gehen wir von Monitoringdaten aus. Dabei enthält ein Paket Informationen über einen Knoten im Netz. Dazu gehören dessen IP-Adresse, Taktrate (in GHz) und Auslastung (in %) der CPU. Ausserdem sollten noch für für jede vom Knoten ausgehende Verbindung (Link), der eineindeutige Name der Verbindung, die Bandbreite (in Kb/s) und wiederum die Auslastung angegeben werden (siehe Abbildung 3.3 für ein Beispiel).

Um veraltete Pakete zu erkennen, muss zusätzlich eine Sequenznummer vorhanden sein. Wofür der Reset-Eintrag ist, wird unter 3.3.2.3 erklärt. Pakete, die nicht diesem Datenformat entsprechen, werden verworfen.

```
<InfoFusionData>
        <reset>false</reset>
        <sequence_number>1002</sequence_number>
        <host_IP_address>129.132.57.86</host_IP_address>
        <cpu_clock_rate>2.8 GHz</cpu_clock_rate>
        <cpu_utilisation>62.3%</cpu_utilisation>
        <links>
            <link>129.132.57.86eth0</link>
            <bandwidth>500 Kb/s</bandwidth>
            <link_utilisation>22.2%</link_utilisation>
            <link>129.132.57.86eth1</link>
            <bandwidth>250 Kb/s</bandwidth>
            <link_utilisation>88.4%</link_utilisatio/>
        </links>
</InfoFusionData>
```

Abbildung 3.3: Beispiel für Dateninhalt eines Paketes

3.3.2.2 Parameter

Wir wissen nun, von welchem Datenformat der Service ausgeht. Welche Daten wird aber das Resultat beinhalten? Diese Frage soll an dieser Stelle anhand der Konfigurationsparameter des Services beantwortet werden.

Die grundsätzliche Idee ist, dass wir nur an Extremwerten interessiert sind. Konkret wollen wir wissen, welcher Knoten im Netz die maximale und / oder die minimale CPU-Taktrate, -Auslastung bzw. die maximale / minimale ungenützte Rechenleistung besitzt. Dasselbe gilt für die Verbindungen im Netz, wo das Interesse dem Link mit maximaler / minimaler Bandbreite, Auslastung bzw. ungenützter Bandbreite gilt.

Zusätzlich soll man über geeignete Parameter angeben können, an welchen Knoten und Verbindungen man prinzipiell interessiert ist. Folglich werden die vorhin erwähnten Extrema bzgl. dieser Knoten und Verbindungen berechnet, erhaltene Informationen über andere Knoten und Verbindungen sollten keinen Einfluss auf das Resultat haben. Tabelle 3.1 gibt einen Überblick über die angesprochenen und zusätzliche Parameter des Services.

PARAMETER	ZWECK
netlink_family	Selektiert das Kernel-Modul, mit dem wir kommunizieren wollen. ToNetlink entspricht der netlink_family 7.
netlink_group	Spezifiziert die Multicast-Gruppe (nur relevant falls netlink_pid nicht spezifiert oder gleich Null ist).
netlink_pid	Optional. Falls angegeben und ungleich Null, werden Unicast-Nachrichten empfangen, die an die spezifierte PID adressiert sind. Sonst werden nur Multicast-Nachrichten empfangen.
hosts	Optional. Gibt anhand von IP-Adressen an, an welchen Knoten (Hosts) wir interessiert sind. Falls nicht spezifiert oder gleich „all", gilt das Interesse allen Hosts.
clockRateXtrema cpuUtilXtrema cpuProdXtrema	Dient der Angabe, an welchen Extrema (Maximum, Minimum oder beide) der CPU-Taktraten, -Auslastungen bzw. der Produkte (1 - Auslastung/100) * CPU-Taktrate (gleich ungenutzte CPU-Rechenleistung) der unter hosts angegebenen Knoten wir interessiert sind.
links	Optional. Gibt an, an welchen Verbindungen wir interessiert sind. Falls nicht spezifiert oder gleich „all", gilt das Interesse allen Verbindungen.
bandwidthXtrema linkUtilXtrema linkProdXtrema	Analog zu clockRate-/cpuUtil-/cpuProdXtrema für Bandbreiten, Auslastungen bzw. Produkte (1 - Auslastung/100) * Bandbreite (gleich ungenutzte Bandbreite) von unter links angegebenen Verbindungen .
destHost destPort	Zielknoten (IP-Adresse und Portnummer), an den die aggregierten Resultate geschickt werden sollen.
interval	Zeitspanne (in Millisekunden) während der Daten gesammelt werden, bevor das nächste aggregierte Resultat versendet wird.

Tabelle 3.1: Service-Parameter

3.3.2.3 Algorithmus

Nachdem besprochen wurde, von welchen Daten ausgegangen wird und wir auch wissen, welche Informationen aggregierte Resultate enthalten sollen, fragt es sich, welche Berechnungen durchgeführt werden müssen, um den Übergang von ersterem zu letzterem zu meistern.

Den Einstiegspunkt in den Algorithmus bildet eine Methode (processPacket (int)) der InfoFusion-Komponente, welche vom Adapter JNetlink via JNI aufgerufen wird, sobald über den Netlink-Socket der Inhalt eines Paketes in ein Java-Bytearray eingelesen wurde.

Die allererste Aufgabe besteht darin, die erhaltenen Daten zu parsen. Dazu nutzen wir aus, dass die einzelnen Tokens im Datenpuffer durch die in Markup-Languages üblichen Separatoren (< bzw. >) voneinander getrennt sind (siehe Abbildung 3.3). Sobald wir beim durchiterieren des Puffers auf einen Separator stossen, wissen wir, dass wir das Ende eines Tokens bzw. den Anfang des nächsten erreicht haben. Da XML- und HTML-Dokumente eine ähnliche Struktur haben, wurde zu Testzwecken versucht, diese Parse-Methode auch auf HTML-Dokumente anzuwenden, was erfolgreich verlief. Wir speichern die so extrahierten Tokens in einem Java-Vector namens tokens ab, um später darauf zugreifen zu können.

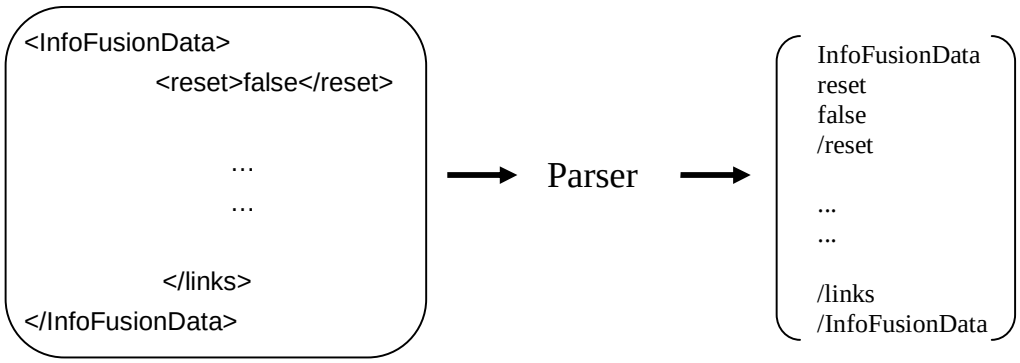

Abbildung 3.4: Parsen der erhaltenen Daten

Nach dem parsen des Pufferinhalts kann man die einzelnen Einträge analysieren. Priorität hat dabei host_IP_address, weil damit überprüft werden kann, ob wir schon einmal ein Paket mit Informationen zum entsprechenden Knoten erhalten haben. Dazu machen wir von einem Java-Vector, im Folgenden collectedHosts genannt, Gebrauch. Darin speichern wir für jeden Knoten, zu dem wir Informationsdaten erhalten haben, u.a dessen IP-Adresse ab. Ausserdem wird noch festgehalten, ob der Knoten oder die Links, an die er angeschlossen ist, relevant sind. Was in diesem Zusammenhang (ir)relevant heisst, folgt gleich im Anschluss. Weiter wird für jeden Knoten die Sequenznummer des zuletzt erhaltenen Paketes, welches Daten zu diesem Knoten enthielt, gespeichert.

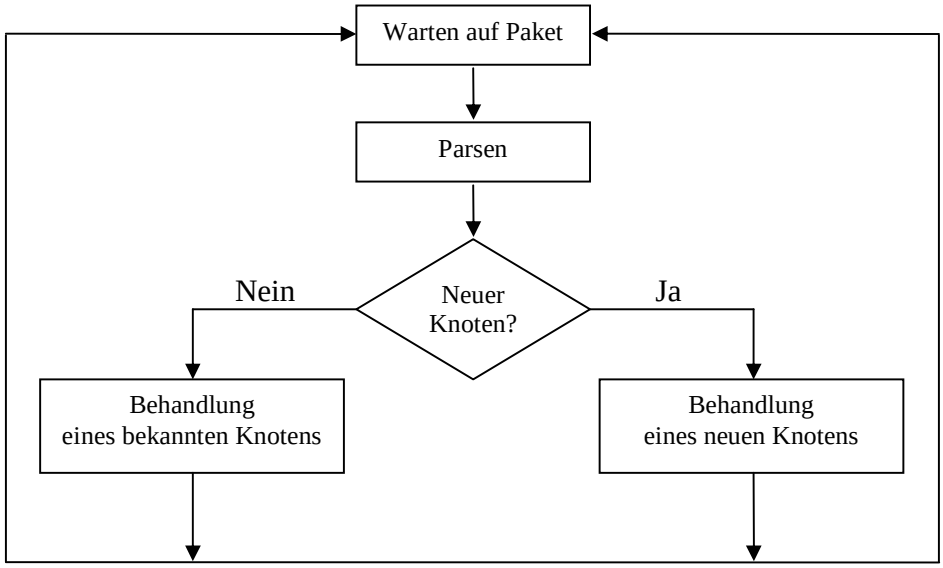

Abbildung 3.5: 1. Teil des Algorithmus

Handelt es sich beim Knoten um einen neuen, nicht in collectedHosts enthaltenen, muss überprüft werden, ob wir an diesem Knoten oder den zugehörigen Verbindungen, interessiert sind. Dies kann leicht mit Hilfe der Service-Parameter hosts bzw. links (siehe Tabelle 3.1) erfolgen. Trifft beides nicht zu, steht fest, dass die erhaltenen Daten irrelevant sind. Folglich muss der Knoten in collectedHosts für zukünftige Berechnungen als irrelevant abgespeichert werden. Die Abarbeitung des Paketes ist hiermit abgeschlossen. Tritt hingegen das Gegenteil ein, d.h. wir sind an diesem Knoten oder den zugehörigen Links interessiert, werden die entsprechenden Tokens (cpu_clock_rate, cpu_utilisation etc.) aus dem Java-Vector tokens ausgelesen, um das aggregierte Resultat gegebenenfalls upzudaten (die Berechnung des Resultates wird später behandelt). Ausserdem bleibt der Knoten mit denjenigen Verbindungen, welche von Interesse sind, als relevant abgespeichert.

Der andere Fall, wo wir den Knoten in collectedHosts auffinden, setllt sich etwas komplizierter. Zuerst muss, ebenfalls unter Einbezug von collectedHosts, geschaut werden, ob der Knoten oder an ihn anliegende Verbindungen von Interesse sind (das wird ja in collectedHosts abgespeichert). Trifft beides nicht zu, brauchen wir das Paket nicht weiter zu verarbeiten (da es nur Daten enthält, die nicht von Interesse sind) und können uns dem nächsten zuwenden.

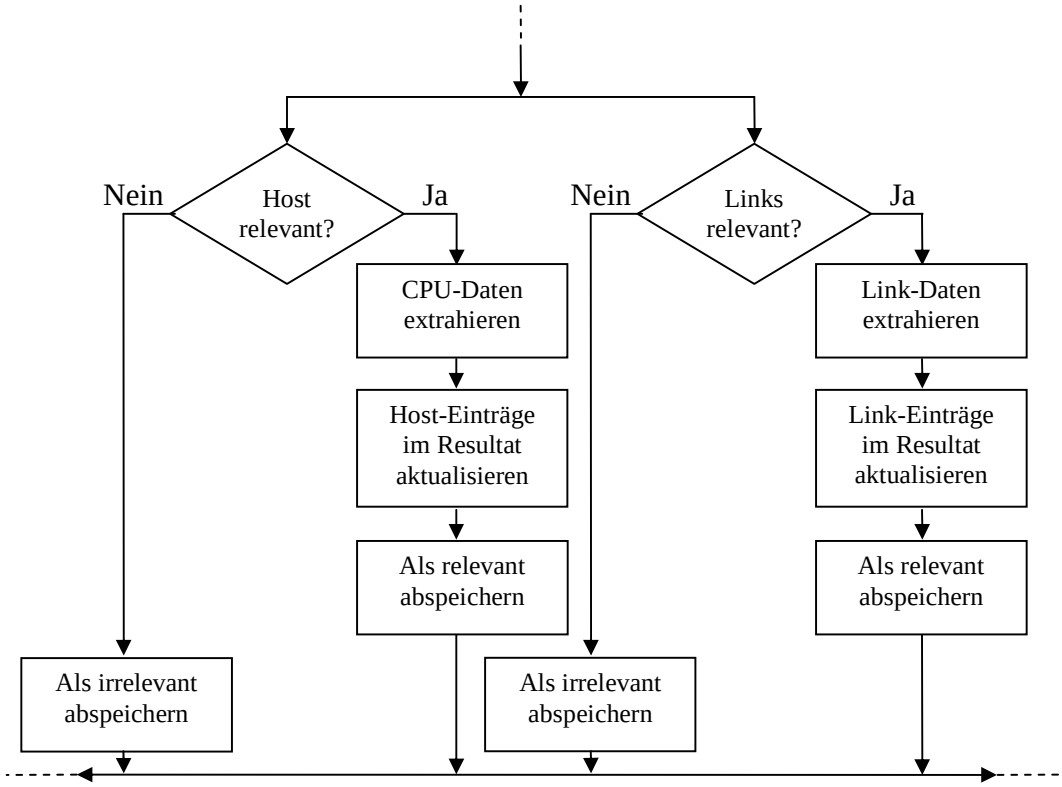

Abbildung 3.6: Behandlung eines neuen Knotens

Ansonsten kommen nun Reset-Eintrag und die Sequenznummer ins Spiel. Die Idee hinter dem Resetflag ist folgende: Angenommen die Datenquelle X generiert Monitoring-Pakete gemäss Abbildung 3.3 zu einem bestimmten Netzwerkknoten. Jedes Datenpaket enthält eine Sequenznummer, so dass wir immer die höchsterhaltene Sequenznummer abspeichern können, um veraltete von neueren Paketen unterscheiden zu können. Was passiert aber, wenn die Datenquelle X abstürzt und dann wieder ihre Tätigkeit aufnimmt? Wenn sie sich nicht durch geeignete Massnahmen an die zuletzt gesendete Sequenznummer erinnern kann, wird sie bei der Durchnummerierung der zu versendenden Pakete wieder von vorne beginnen. Offensichtlich stempelt der InfoFusion-Service die so erhaltenen Pakete als veraltet ab, bis wieder eine Sequenznummer erreicht wird, die diejenige vor dem Absturz der Quelle abgespeicherte übertrifft. Um dem entgegenzuwirken, kann der Reset-Eintrag auf true gesetzt werden, womit die Sequenznummer nicht überprüft wird.

Nachdem wir uns vergewissert haben, dass der Knoten oder an ihn anliegende Verbindungen relevant sind, gilt es also, das Resetflag zu prüfen. Ist dieses gesetzt, dürfen wir die Sequenznummer nicht berücksichtigen. Ansonsten vergleichen wir die soeben erhaltene mit der zum Knoten gespeicherten Sequenznummer. Ist sie kleiner, folgt, dass das Paket veraltet ist, wir können es demzufolge verwerfen. Sonst muss man sich einerseits die neue Sequenznummer merken, andererseits müssen die neuen Werte wie CPU-Taktrate, -Auslastung etc. aus dem tokens-Java-Vector ausgelesen und abgespeichert werden. Zu guter letzt muss natürlich das Resultat gegebenenfalls aktualisiert werden. Letzteres wird im nächsten Abschnitt erklärt.

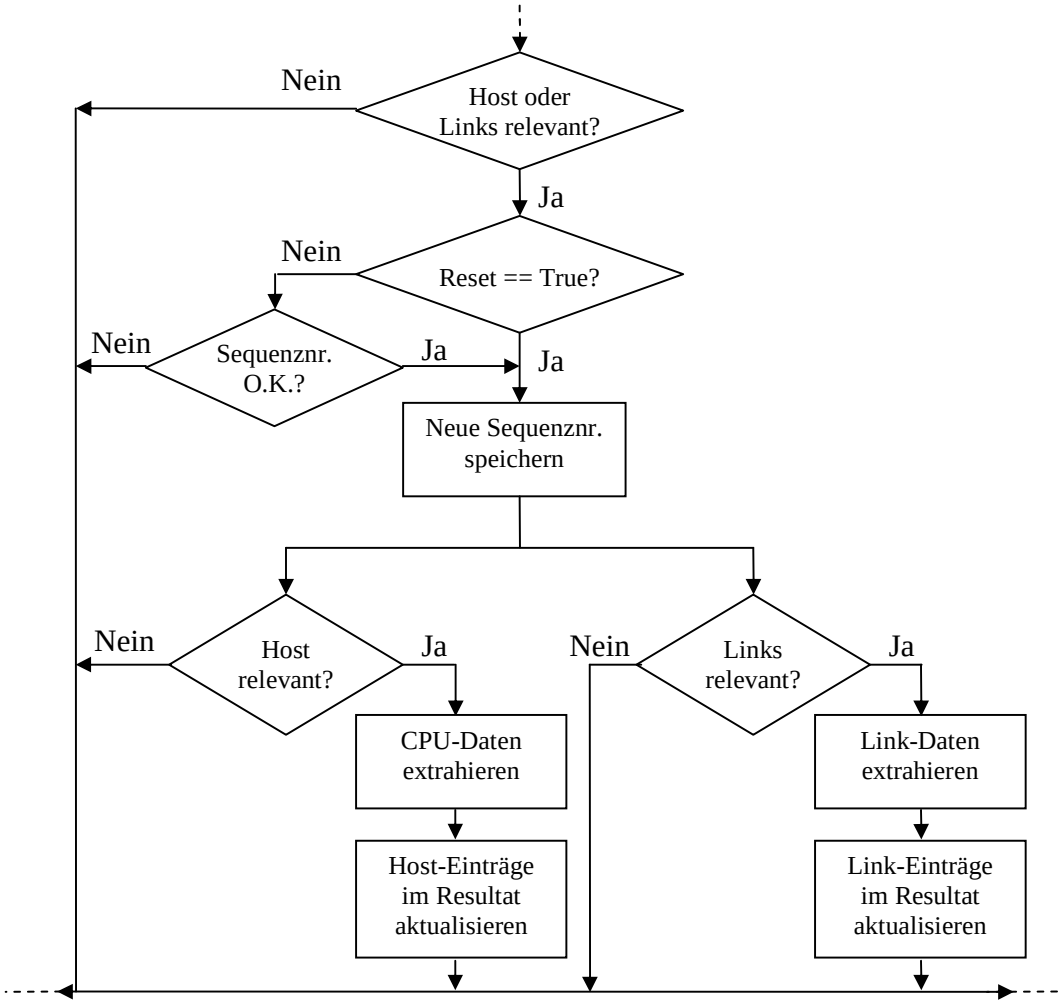

Abbildung 3.7: Behandlung eines bekannten Knotens

Nun können wir uns der (Neu-)Berechnung des Resultates zuwenden. Im Resultat soll gespeichert werden, welcher Knoten die maximale / minimale CPU-Taktrate (hängt davon ab, woran wir interessiert sind, siehe dazu Tabelle 3.1) hat, dasselbe für die CPU-Auslastung und die ungenutzte Rechenleistung. Ausserdem soll darin festgehalten werden, welche Verbindung die grösste / kleinste Bandbreite hat, wiederum dasselbe für die Auslastung und die ungenutzte Bandbreite. Abbildung 3.8 zeigt ein Beispiel. Das Resultat besteht also im Grunde genommen aus zwei voneinander unabhängigen Teilen: Den Extrema zu CPU-Daten und denjenigen zu Verbindungen. Unabhängig deswegen, weil z.B. ein Knoten der die CPU mit maximaler Taktrate besitzt nicht zwingendermassen auch an eine Verbindung mit maximaler Bandbreite, Auslastung etc. angeschlossen sein muss. Wegen dieser Unabhängigkeit wird bei Ankunft eines Paketes unterschieden, ob die darin enthaltenen CPU-Informationen oder die Daten zu den Verbindungen relevant sind. Folglich wird dann auch nur der entsprechende Teil des Resultates aktualisiert, siehe dazu auch Abbildungen 3.6 bzw. 3.7. Hierfür stehen zwei Methoden (updateCPUExtrema(String) bzw. updateLinkExtrema(String)) zur Verfügung. Der Parameterstring gibt an, ob das zuletzt erhaltene Paket Informationen zu einem neuen, noch unbekannten Knoten und dessen Links enthielt (d.h. der Knoten ist nicht in collectedHosts vorhanden) oder zu einem bekannten, in collectedHosts registrierten.

Maximum of CPU clock rate: 2.8 GHz at host 129.132.57.86
Minimum of CPU utilisation: 19.40% at host 186.8.8.44
Maximum of product (1 - CPU utilisation / 100) * CPU clock rate: 1.74 GHz at host 2.4.8.4
Minimum of product (1 - CPU utilisation / 100) * CPU clock rate: 0.90 GHz at host 5.1.1.4
Maximum of link bandwidth: 500 Kb/s at link 129.132.57.86eth0
Maximum of link utilisation: 88.4% at link 129.132.57.86eth0
Minimum of product (1 - link utilisation / 100) * link bandwidth: 40 Kb/s at link 2.4.8.4eth1

Abbildung 3.8: Beispiel für ein aggregiertes Resultat

Betrachten wir zunächst das Aktualisieren der CPU-Extremwerte (updateCPUExtrema(String)) im Resultat. Hier muss vorerst anhand des Parameters der Methode unterschieden werden, ob es sich um einen "neuen" Knoten handelt. Trifft dies zu, schauen wir in den Service-Parametern (siehe Tabelle 3.1) nach, an welchen Extrema der CPU-Taktrate wir interessiert sind (Maximum, Minimum oder beide). Entsprechend vergleichen wir die Taktrate des Knotens mit der im Resultat abgespeicherten. Wenn wir einen neuen Extremwert haben, wir sind z.B. am Maximum interessiert und der Knoten hat eine höhere Taktrate als der im Resultat

abgespeicherte Knoten, wird im Resultat entsprechend der neue Knoten als „Rekordhalter“ notiert. Falls hingegen kein neuer Extremwert resultiert, bleibt das Resultat unberührt. Dasselbe Prozedere wird für die CPU-Auslastung und die ungenutzte Rechenleistung durchgeführt.

Gibt der Methodenparameter an, dass es sich um einen bekannten Knoten handelt, nennen wir diesen Knoten X, ändert sich ein kleines aber wichtiges Detail. Nehmen wir als Beispiel die CPU-Auslastung und betrachten den Fall, wo wir am Minimum interessiert sind und im Resultat Knoten X selbst als Knoten mit minimaler CPU-Auslastung notiert wurde. Sei weiter die CPU-Auslastung von X neu höher als zur Zeit, wo X als Knoten mit minimaler Auslastung abgespeichert wurde. Dann können wir uns nicht einfach damit begnügen, das Resultat so beizubelassen, denn es könnte sein, dass ein ebenfalls relevanter Knoten Y eine höhere CPU-Auslastung als die damalige, jedoch eine kleinere als die neue Auslastung des Knotens X besitzt. Logischerweise muss deswegen in solchen Fällen durch alle bekannte, als relevant abgespeicherte Knoten iteriert werden, um so das Resultat korrekt zu halten. Die Fälle wo wir am Maximum interessiert sind bzw. im Resultat die Einträge für die CPU-Taktrate und die ungenutzte Rechenleistung aktualisieren müssen, verhalten sich analog. Ausser diesem speziellen Fall (in Abbildung 3.9 ist unter Spezialfall dieser gemeint) gibt es sonst keine Unterschiede zum Prozedere bei einem „neuen“ Knoten.

Die Berechnung desjenigen Teils des Resultates, welches sich mit den Extremwerten von Verbindungen befasst, verhält sich komplett analog zur Berechnung der CPU-Extrema. Für das Maximum der Bandbreite wird z.B. die Bandbreite des im Resultat abgespeicherten Links mit der maximalen Bandbreite, aggregiert über alle im Paket enthaltene, relevante Verbindungen, verglichen usw.

Um schlussendlich das Resultat dem druch Service-Parameter spezifizierten Zielknoten zuzusenden (siehe Tabelle 3.1), braucht es einen separaten Thread, der alle interval Millisekunden (siehe wiederum Tabelle 3.1) das bis dahin aggregierte Resultat verschickt. Wenn man auf einen separaten Thread verzichten würde, könnte man offensichtlich jeweils nur dann überprüfen, ob das Intervall abgelaufen ist, wenn wieder ein Paket angekommen ist, sprich der JNetlink-Adapter die processPacket-Methode aufgerufen hat. Dies steht jedoch in keinem Zusammenhang zum Intervall, weshalb diese Lösung nicht anwendbar ist.

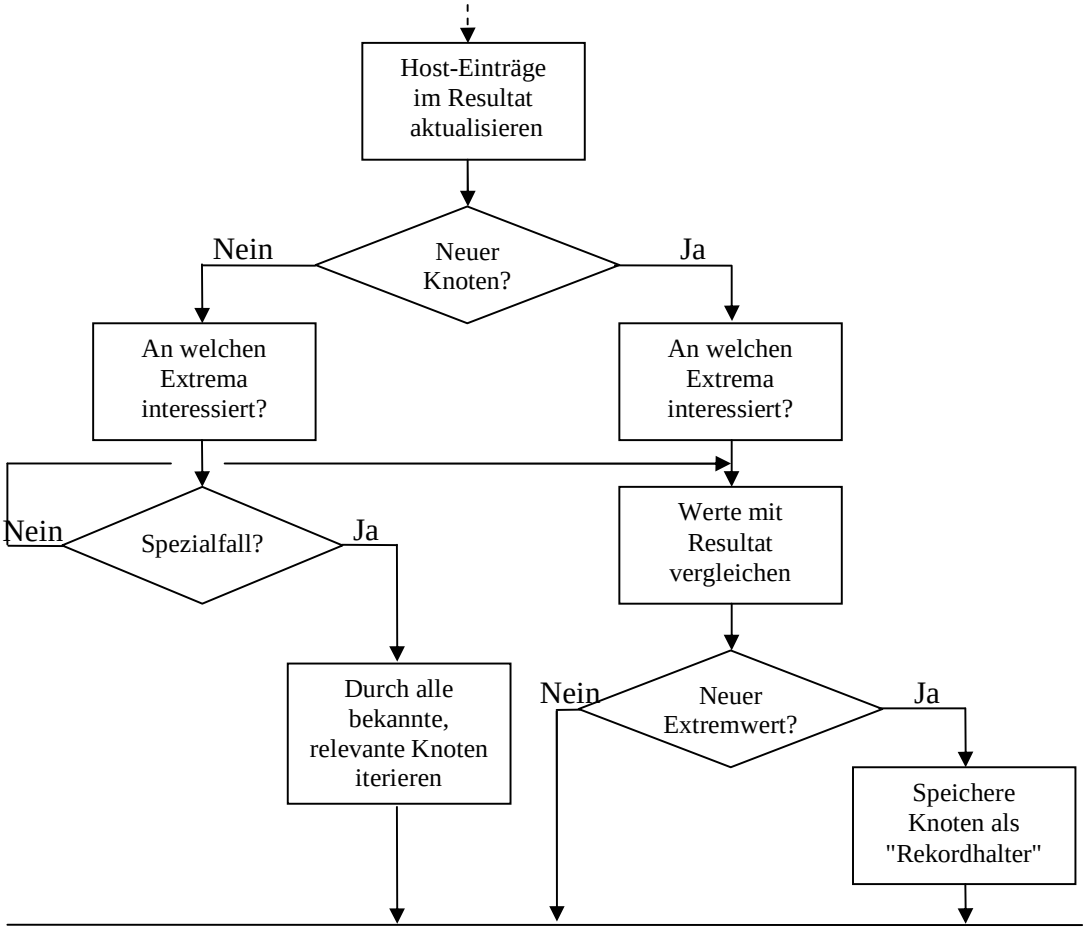

Abbildung 3.9: Aktualisierung des Host-Teils des Resultates

3.4 Zusätzliche Komponenten

Um den Service zu testen, mussten noch zwei zusätzliche Java-Applikationen implementiert werden, die hier kurz beschrieben werden sollen.

3.4.1 Packet-Generator

Damit der Information-Fusion Service Pakete zu Gesicht bekommt, wurde eine kleine Applikation entwickelt, welche UDP-Pakete generiert, deren Datenformat demjenigen in Abbildung 3.3 entspricht. Anhand von Kommandozeilen-Parametern wird angegeben, mit welchen Werten die entsprechenden Einträge (z.B. host_IP_address) gefüllt werden sollen. Weiter kann man die IP-Adresse und den Port des Rechners angeben, an den die Pakete geschickt werden sollen. Schlussendlich gibt ein weiterer Parameter an, mit welcher Rate Pakete zu generieren sind. Für die genaue Verwendung dieser Applikation sei auf Anhang A verwiesen.

3.4.2 Empfänger

Da die aggregierten Resultate an einen bestimmten Zielknoten geschickt werden, macht es Sinn, auf dem entsprechenden Knoten ein Programm laufen zu lassen, welches solche Pakete entgegenimmt. Dabei kann wiederum über Kommandozeilen-Parameter angegeben werden, an welchem Port gehorcht werden soll. Falls kein Parameter übergeben wurde, wird eine hartkodierte Portnummer verwendet.

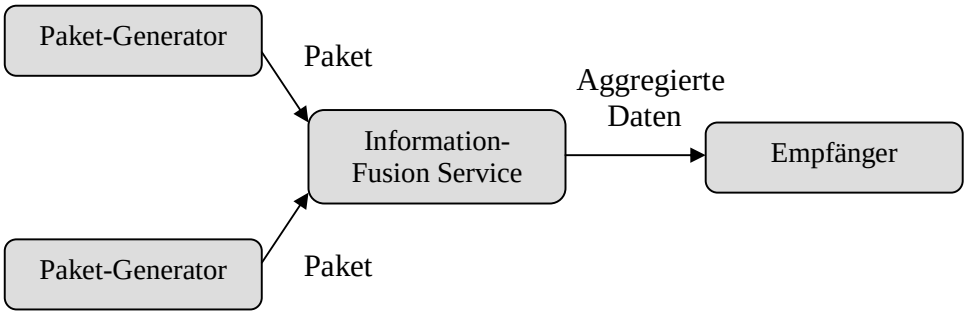

Abbildung 3.10: Alle verwendeten Komponenten

Kapitel 4

Fazit

Dieses Kapitel dient einer kleinen Zusammenfassung der erreichten Ziele und der gesammelten Eindrücke bzw. Erfahrungen. Ausserdem soll auch auf Vorschläge für Verbesserungen und zukünftige Arbeiten eingegangen werden.

4.1 Erreichte Ziele

Der implementierte Information-Fusion Service setzt sich sowohl aus Komponenten zusammen, die in der Click EE ausgeführt werden, wie auch aus solchen, die in der Chameleon Java EE zur Ausführung kommen. Damit diese Komponenten über EE-Grenzen hinweg miteinander kommunizieren können, wurden sogenannte Adapter-Module implementiert, die die Verwendung von Netlink für die Inter-EE Kommunikation von der Click EE zur Chameleon Java EE erlauben (nicht aber in die andere Richtung, siehe dazu Abschnitt 4.3). Damit diese und nicht die ursprünglichen Proc-Filesystem-Adapter verwendet werden, wurden die EE-Konfiguratoren entsprechend modifiziert. Die Node Description musste ebenfalls abgeändert werden, um die im Service verwendeten Inter-EE Verbindungen zu unterstützen.

Trotz anfänglichen Schwierigkeiten ist es auch gelungen, den Service zu deployen. Dafür mussten die Service-Deskriptoren geschrieben werden, welche es der Service Creation Engine ermöglichen, die für den Service benötigten Module zu identifizieren und zu installieren.

4.2 Eindrücke und Erfahrungen

Diese Arbeit bot die Gelegenheit, sich ausgiebig mit Click und Chameleon auseinanderzusetzen. Beide haben einen bleibenden Eindruck hinterlassen. Einerseits ist Click so beeindruckend, weil es relativ einfach zu verstehen und zu benutzen ist.

Aus einzelnen, simplen Software-Komponenten können beliebig komplexe Konfigurationen erstellt werden, die das verhalten des Routers bestimmen. Chameleon andererseits erweitert einen Click Router so, dass darauf "beliebige" Services deployed werden können. Um den Information-Fusion Service zu deployen war es nötig, recht detailliert auf die einzelnen Komponenten der SCE einzugehen. Dabei ist aufgefallen, wie komplex aber trotzdem übersichtlich diese implementiert wurde, deshalb sei hier ein grosses Lob an die Entwickler Andreas Moser und Roman Hoog Antink ausgesprochen.

4.3 Vorschläge für Verbesserungen und zukünftige Arbeiten

Die aktuelle Version der InfoFusion-Komponente erlaubt es offenbar nicht, mehrere Instanzen hintereinanderzuschalten, so dass mehrfache Fusions erzwungen werden. Das liegt darin, dass das Output- ungleich dem Input-Format der Pakete ist. Wenn man die Komponente leicht abändert, so dass Output-Pakete das gleiche Datenformat wie Input-Pakete haben, kann das Problem also recht leicht behoben werden (aus Zeitgründen konnte dies nicht mehr vorgenommen werden).

Das schon häufig angesprochene FromNetlink-Element, welches die Inter-EE Kommunikation von der Chameleon Java EE zur Click EE ermöglichen soll, könnte sicherlich mit relativ wenig Aufwand "repariert" werden. Dadurch wäre es möglich, Services zu implementieren, welche Netlink für beide Kommunikations-Richtungen verwenden. Es scheint jedoch schwierig zu sein, Dokumentationsmaterial zum Netlink Kernel-API zu finden.

Zur Zeit ist es so, dass man in den EE-Konfiguratoren hartcodiert festlegt, welche Adapter gewünscht sind. Man könnte die SCE so modifizieren, dass es möglich ist, z.B. in der Node oder Service Description anzugegeben, welche Adapter-Module (Proc-Filesystem oder Netlink) die SCE verwenden soll.

Literatur

[1] Matthias Bossardt, Roman Hoog Antink, Andreas Moser und Bernhard Plattner. Chameleon: Realizing automatic service composition for extensible active routers. In Proceedings of the Fifth Annual International Working Conference on Active Networks IWAN, number 2546 in Lecture Notes in Computer Science, Kyoto, Japan, Dezember 2003. Springer Verlag.

[2] Roman Hoog Antink und Andreas Moser. Service composition for active networks. Masterarbeit, ETH Zürich, Schweiz, 2003.

[3] Reto Zürcher und Andreas Mühlemann. Pattern-based service deployment in active networks. Semesterarbeit, ETH Zürich, Schweiz, 2003.

[4] The Click Modular Router Project
http://www.pdos.lcs.mit.edu/click/

[5] Gowri Dhandapani und Anupama Sundaresan. Netlink sockets – Overview. University of Kansas, Lawrence, U.S.A., September 1999.

[6] Java Native Interface Tutorial
http://java.sun.com/docs/books/tutorial/native1.1/

Anhang A

User Guide

1. Für die Installation von Click und Chameleon sei auf Appendix A von [2] verwiesen.

2. /SA_JulioPerez/jeeNetlink/util/HostConstants.java an eigene Bedürfnisse anpassen und nach /an/java/ch/ethz/ee/tik/chameleon/util/ der Installation unter 1. kopieren.
   /SA_JulioPerez/jeeNetlink/click/Configurator-click.java und Configurator_jee.java nach /an/java/ch/ethz/ee/tik/chameleon/sce/ der Installation unter 1. kopieren.
   Um danach diese Java-Files zu kompilieren, verwende man das Makefile in /an/java der Installation unter 1. (einfach make ausführen).

3. In /SA_JulioPerez/jeeNetlink/demo/run1.sh SERVICE_SRC und CODE_SERVER an eigene Bedürfnisse anpassen.
   run1.sh nach /an/java/ der Installation unter 1. kopieren.

4. /SA_JulioPerez/jeeNetlink/click/tonetlink.cc und tonetlink.hh nach /archive/click/elements/linuxmodule/ der Installation unter 1. kopieren

5. tonetlink-Element installieren (siehe dazu FAQ in /archive/click/)

   - Ins /archive/click/-Verzeichnis wechseln
   - make elemlist
   - make install

   Normalerweise sollte nach einer Installation eines neuen Click-Elementes der Computer rebooted werden, da erst dann das Element wirklich installiert ist.

6. In /SA_JulioPerez/jeeNetlink/ mit make die InfoFusion-Komponente, den Paket-Generator und den Empfänger falls nötig kompilieren.
   Anschliessend make install ausführen.

7. Bei Ausführung von run1.sh werden folgende Programme gestartet:

   - Service Creation Engine
   - Code Server
   - Java EE
   - Service Server
   - Click Configuration Server
   - Demux Configurator

   Jetzt können Service-Requests geschickt werden

8. In /SA_JulioPerez/jeeNetlink/demo/run1.sh SERVICE_SRC und REQUESTER an eigene Bedürfnisse anpassen.

   Die Ausführung von run2.sh bewirkt folgendes:

   - Empfänger (ResultReceiver) wird gestartet
   - Service-Request (DemoServiceReq.xml) für den Information-Fusion Service wird versendet (ServiceRequester)
   - Zwei Paket-Generatoren (StartUDP) werden gestartet

   IP-Adressen müssen zwingend an eigene Bedürfnisse angepasst werden, d.h.:

   - In /SA_JulioPerez/jeeNetlink/xml/DemoServiceReq.xml die Parameter destHost (und destPort) anpassen
   - Falls in DemoServiceReq.xml destPort nicht auf 10000 gesetzt wurde (Defaultport), muss in run2.sh dem ResultReceiver der entsprechende Port als Argument übergeben werden (also z.B. jeeNetlink.ResultReceiver 20000)
   - In run2.sh den Paket-Generatoren die IP-Adressen des Chameleon-Knotens (auf dem der Information-Fusion Service angeboten wird) als Destination angeben (statt 129.132.57.127 z.B. 1.1.1.1). Falls derselbe Rechner für die Paket-Generierung und als Chameleon-Knoten dient, sollte eine Broadcast-Adresse verwendet werden.
   - Eventuell in InfoFusionService.xml Netfilter-Rule abändern oder weitere hinzufügen

Anhang B

XML-Dateien

Anhang B: XML-Dateien

```
<?xml version="1.0" encoding="UTF-8"?>
<NODE_DESCRIPTION xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Chameleon\XML\chameleon.xsd">
    <!-- configurator: name of java class configuring this EE
         name: name of this EE, corresponds to service descriptions -->
    <xsi:OS xsi:name="Linux" xsi:version="2.4.20"/>
    <xsi:OS xsi:name="Linux" xsi:version="2.4.19"/>
    <xsi:OS xsi:name="Linux" xsi:version="2.4.18"/>
    <xsi:OS xsi:name="AS400" xsi:version="23.4.56.pre12 patchlevel 5"/>
    <xsi:EE xsi:name="JEE" xsi:configurator="jee" xsi:version="1">
        <xsi:PORT xsi:type="pull_in"/>
        <xsi:PORT xsi:type="push_in"/>
        <xsi:PORT xsi:type="pull_out"/>
        <xsi:PORT xsi:type="push_out"/>
    </xsi:EE>
    <xsi:EE xsi:name="CLICK" xsi:configurator="click" xsi:version="1"
            xsi:entry_module="FromNetfilter" xsi:exit_module="ToDevice"
            xsi:demux_target="-j CLICK --click-wire ">
        <xsi:PORT xsi:type="pull_in"/>
        <xsi:PORT xsi:type="push_in"/>
        <xsi:PORT xsi:type="pull_out"/>
        <xsi:PORT xsi:type="push_out"/>
        <xsi:PORT xsi:type="agnostic_in"/>
        <xsi:PORT xsi:type="agnostic_out"/>
        <xsi:PORT xsi:type="procfs_in" xsi:flag="optional"/>
        <xsi:PORT xsi:type="procfs_out" xsi:flag="optional"/>
    </xsi:EE>
    <xsi:EE_CONNECTIONS xsi:fromEE="CLICK" xsi:toEE="JEE">
        <xsi:CONNECTION xsi:fromPort="procfs_out" xsi:toPort="push_in"/>
       <xsi:CONNECTION xsi:fromPort="agnostic_out" xsi:toPort="push_in"/>
    </xsi:EE_CONNECTIONS>
    <xsi:EE_CONNECTIONS xsi:fromEE="JEE" xsi:toEE="CLICK">
        <xsi:CONNECTION xsi:toPort="procfs_in" xsi:fromPort="push_out"/>
    </xsi:EE_CONNECTIONS>
    <xsi:EE_CONNECTIONS xsi:fromEE="CLICK" xsi:toEE="CLICK">
        <xsi:CONNECTION xsi:fromPort="pull_out" xsi:toPort="pull_in"/>
        <xsi:CONNECTION xsi:fromPort="push_out" xsi:toPort="push_in"/>
        <xsi:CONNECTION xsi:fromPort="pull_out"xsi:toPort="agnostic_in"/>
        <xsi:CONNECTION xsi:fromPort="push_out" xsi:toPort="agnostic_in"/>
        <xsi:CONNECTION xsi:fromPort="agnostic_out" xsi:toPort="agnostic_in"/>
        <xsi:CONNECTION xsi:fromPort="agnostic_out" xsi:toPort="push_in"/>
        <xsi:CONNECTION xsi:fromPort="agnostic_out" xsi:toPort="pull_in"/>
    </xsi:EE_CONNECTIONS>
    <xsi:EE_CONNECTIONS xsi:fromEE="JEE" xsi:toEE="JEE">
        <xsi:CONNECTION xsi:fromPort="pull_out" xsi:toPort="pull_in"/>
        <xsi:CONNECTION xsi:fromPort="push_out" xsi:toPort="push_in"/>
    </xsi:EE_CONNECTIONS>
</NODE_DESCRIPTION>
```

Abbildung B.1: Node Description

Anhang B: XML-Dateien

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICE_LIST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Chameleon\XML\chameleon.xsd">
<SERVICE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="SPECIFICATION">
    <DESCRIPTION xsi:servicename="InfoFusionService" xsi:option="normal">
        <PROVIDER>ETH</PROVIDER>
        <VERSION>2.1</VERSION>
    </DESCRIPTION>
    <PORTS>
        <IN_PORT xsi:type="push_in"/>
    </PORTS>
    <DEMUX_RULE>-p udp --dport 20000</DEMUX_RULE>
    <PARAM xsi:name="netlink_family"/>
    <PARAM xsi:name="netlink_group"/>
    <PARAM xsi:name="PID"/>
    <PARAM xsi:name="netlink_pid"/>
    <PARAM xsi:name="hosts"/>
    <PARAM xsi:name="clockRateXtrema"/>
    <PARAM xsi:name="cpuUtilXtrema"/>
    <PARAM xsi:name="cpuProdXtrema"/>
    <PARAM xsi:name="links"/>
    <PARAM xsi:name="bandwidthXtrema"/>
    <PARAM xsi:name="linkUtilXtrema"/>
    <PARAM xsi:name="linkProdXtrema"/>
    <PARAM xsi:name="destHost"/>
    <PARAM xsi:name="destPort"/>
    <PARAM xsi:name="interval"/>
    <SUB_SERVICE xsi:name="Strip" xsi:instance_name="strip"/>
    <SUB_SERVICE xsi:name="CheckIPHeader"xsi:instance_name="checkIpHeader"/>
    <SUB_SERVICE xsi:name="ToNetlink" xsi:instance_name="toNetlink">
         <PARAM xsi:name="PID"/>
         <PARAM xsi:name="netlink_group"/>
    </SUB_SERVICE>
    <SUB_SERVICE xsi:name="InfoFusion" xsi:instance_name="infoFusion">
         <PARAM xsi:name="netlink_family"/>
         <PARAM xsi:name="netlink_group"/>
         <PARAM xsi:name="netlink_pid"/>
         <PARAM xsi:name="hosts"/>
         <PARAM xsi:name="clockRateXtrema"/>
         <PARAM xsi:name="cpuUtilXtrema"/>
         <PARAM xsi:name="cpuProdXtrema"/>
         <PARAM xsi:name="links"/>
         <PARAM xsi:name="bandwidthXtrema"/>
         <PARAM xsi:name="linkUtilXtrema"/>
         <PARAM xsi:name="linkProdXtrema"/>
         <PARAM xsi:name="destHost"/>
         <PARAM xsi:name="destPort"/>
         <PARAM xsi:name="interval"/>
    </SUB_SERVICE>
```

```
    <TRANSPORT_CONNECTION>
        <SRC_PORT xsi:service_instance="this" xsi:type="push_in"/>
        <DEST_PORT xsi:service_instance="strip" xsi:type="agnostic_in"/>
    </TRANSPORT_CONNECTION>
    <TRANSPORT_CONNECTION>
        <SRC_PORT xsi:service_instance="strip" xsi:type="agnostic_out"/>
        <DEST_PORT xsi:service_instance="checkIpHeader" xsi:type="agnostic_in"/>
    </TRANSPORT_CONNECTION>
    <TRANSPORT_CONNECTION>
        <SRC_PORT xsi:service_instance="checkIpHeader" xsi:type="agnostic_out"/>
        <DEST_PORT xsi:service_instance="infoFusion" xsi:type="push_in"/>
    </TRANSPORT_CONNECTION>
</SERVICE>
</SERVICE_LIST>
```

Abbildung B.2: XML-Beschreibung des Services

Anhang B: XML-Dateien

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICE_LIST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Chameleon\XML\chameleon.xsd">
<SERVICE xsi:type="IMPLEMENTATION">
    <DESCRIPTION xsi:servicename="FromNetfilter" xsi:option="normal">
        <PROVIDER>ETH</PROVIDER>
        <VERSION>2.1</VERSION>
    </DESCRIPTION>
    <ENVIRONMENT>
        <OS>
            <NAME>Linux</NAME>
            <VERSION>2.4.20</VERSION>
        </OS>
        <EE>
            <NAME>CLICK</NAME>
            <VERSION>1</VERSION>
        </EE>
    </ENVIRONMENT>
    <PORTS>
        <OUT_PORT xsi:type="push_out"/>
    </PORTS>
</SERVICE>
<SERVICE xsi:type="IMPLEMENTATION">
    <DESCRIPTION xsi:servicename="Strip" xsi:option="normal">
        <PROVIDER>ETH</PROVIDER>
        <VERSION>2.1</VERSION>
    </DESCRIPTION>
    <ENVIRONMENT>
        <OS>
            <NAME>Linux</NAME>
            <VERSION>2.4.20</VERSION>
        </OS>
        <EE>
            <NAME>CLICK</NAME>
            <VERSION>1</VERSION>
        </EE>
    </ENVIRONMENT>
    <PORTS>
        <IN_PORT xsi:type="agnostic_in"/>
        <OUT_PORT xsi:type="agnostic_out"/>
    </PORTS>
    <PARAM xsi:name="ethHeaderSize" xsi:value="14"/>
</SERVICE>
```

Anhang B: XML-Dateien

```
<SERVICE xsi:type="IMPLEMENTATION">
    <DESCRIPTION xsi:servicename="CheckIPHeader" xsi:option="normal">
        <PROVIDER>ETH</PROVIDER>
        <VERSION>2.1</VERSION>
    </DESCRIPTION>
    <ENVIRONMENT>
        <OS>
            <NAME>Linux</NAME>
            <VERSION>2.4.20</VERSION>
        </OS>
        <EE>
            <NAME>CLICK</NAME>
            <VERSION>1</VERSION>
        </EE>
    </ENVIRONMENT>
    <PORTS>
        <IN_PORT xsi:type="agnostic_in"/>
        <OUT_PORT xsi:type="agnostic_out"/>
    </PORTS>
</SERVICE>
<SERVICE xsi:type="IMPLEMENTATION">
    <DESCRIPTION xsi:servicename="ToNetlink" xsi:option="normal">
        <PROVIDER>ETH</PROVIDER>
        <VERSION>2.1</VERSION>
    </DESCRIPTION>
    <ENVIRONMENT>
        <OS>
            <NAME>Linux</NAME>
            <VERSION>2.4.20</VERSION>
        </OS>
        <EE>
            <NAME>CLICK</NAME>
            <VERSION>1</VERSION>
        </EE>
    </ENVIRONMENT>
    <PORTS>
        <IN_PORT xsi:type="push_in"/>
    </PORTS>
    <PARAM xsi:name="PID"/>
    <PARAM xsi:name="netlink_group"/>
</SERVICE>
</SERVICE_LIST>
```

Abbildung B.3: XML-Beschreibung der Hilfskomponenten

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICE_LIST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Chameleon\XML\chameleon.xsd">
<SERVICE xsi:type="IMPLEMENTATION">
    <DESCRIPTION xsi:servicename="InfoFusion" xsi:option="normal">
        <PROVIDER>ETH</PROVIDER>
        <VERSION>2.1</VERSION>
    </DESCRIPTION>
    <ENVIRONMENT>
        <OS>
            <NAME>Linux</NAME>
            <VERSION>2.4.20</VERSION>
        </OS>
        <EE>
            <NAME>JEE</NAME>
            <VERSION>1</VERSION>
        </EE>
    </ENVIRONMENT>
       <CODE_LOCATION>InfoFusion.class</CODE_LOCATION>
    <PORTS>
        <IN_PORT xsi:type="push_in"/>
    </PORTS>
    <PARAM xsi:name="netlink_family"/>
    <PARAM xsi:name="netlink_group"/>
    <PARAM xsi:name="netlink_pid"/>
    <PARAM xsi:name="hosts"/>
    <PARAM xsi:name="clockRateXtrema"/>
    <PARAM xsi:name="cpuUtilXtrema"/>
    <PARAM xsi:name="cpuProdXtrema"/>
    <PARAM xsi:name="links"/>
    <PARAM xsi:name="bandwidthXtrema"/>
    <PARAM xsi:name="linkUtilXtrema"/>
    <PARAM xsi:name="linkProdXtrema"/>
    <PARAM xsi:name="destHost"/>
    <PARAM xsi:name="destPort"/>
    <PARAM xsi:name="interval"/>
</SERVICE>
</SERVICE_LIST>
```

Abbildung B.4: XML-Beschreibung der Information-Fusion Komponente

Anhang B: XML-Dateien

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Chameleon\XML\chameleon.xsd">
   <DESCRIPTION xsi:servicename="InfoFusionService" xsi:option="ultracool">
      <PROVIDER>ETH</PROVIDER>
      <VERSION>0</VERSION>
   </DESCRIPTION>
   <PARAM xsi:name="netlink_family" xsi:value="7"/>
   <PARAM xsi:name="netlink_group" xsi:value="1"/>
   <PARAM xsi:name="PID" xsi:value="pid 2"/>
   <PARAM xsi:name="netlink_pid" xsi:value="2"/>
   <PARAM xsi:name="hosts" xsi:value="all"/>
   <PARAM xsi:name="clockRateXtrema" xsi:value="max"/>
   <PARAM xsi:name="cpuUtilXtrema" xsi:value="min"/>
   <PARAM xsi:name="cpuProdXtrema" xsi:value="max+min"/>
   <PARAM xsi:name="links" xsi:value="2.2.2.2eth1"/>
   <PARAM xsi:name="bandwidthXtrema" xsi:value="max"/>
   <PARAM xsi:name="linkUtilXtrema" xsi:value="min"/>
   <PARAM xsi:name="linkProdXtrema" xsi:value="max"/>
   <PARAM xsi:name="destHost" xsi:value="129.132.57.127"/>
   <PARAM xsi:name="destPort" xsi:value="10000"/>
   <PARAM xsi:name="interval" xsi:value="2000"/>
</SERVICE>
```

Abbildung B.5: Beispiel für einen Information-Fusion Service-Request

Anhang C

Aufgabenstellung

Anhang C: Aufgabenstellung

Winter 2003/2004

Semesterarbeit

für

Julio Pérez

Supervisor: Matthias Bossardt
Co-Supervisor: Lukas Ruf

Ausgabe: 27.10.2003
Abgabe: 13.2.2004

Active Network Service Using Netlink

---

1 Introduction

Active networks contain programmable routers, which allow users to dynamically install programs. Such programs are typically part of a distributed service deployed in the network. Active routers enable the customization of packet handling in a very flexible way. Possible services include packet filtering, Web caches, load balancing as well as specialized multicasting protocols and video scaling.[1]

An active network node is similar to a classical router in a legacy IP network. In both cases, the main task is to forward data packets. Additionally, an active node allows freely customizing the processing of data packets on the node.

[1] This list is not exhaustive

As a consequence, networks that include active network nodes may offer services in addition to the basic IP forwarding services of classical IP networks. A service is composed of service components that must be deployed on one or more active nodes.

At TIK, we developed a service deployment architecture for active network nodes [2, 1, 3], which allows to identify and deploy service components that must be executed on one specific node. This task is carried out by the Service Creation Engine (SCE).

Service deployment includes installing and configuring software components that perform processing of the data packets. Such components run in execution environments (EE). Many types of execution environments have been developed each of which is usually optimized for a certain type of tasks. Active nodes typically provide more than one to support the whole spectrum of services.

The current implementation features a Java-based and a kernel based EE. From a performance point of view, such a Java-based EE, is not sufficient for transport plane operations. More efficient approaches for this type of task include EEs that execute code in the kernel space. Therefore, we integrated a Click router [4] based EE into our system. Click EE enables simple configuration of the transport plane of a software router. It executes code in the kernel space and provides a variety of service components (Click terminology: elements).

Service components running in different EEs must be able to communicate. Therefore the SCE configures appropriate communication channels between EEs. The most appropriate technology and implementation of a communication channel depends on the EEs that it connects. Currently, communication between the Chameleon Java EE (CJEE) and the Click EE uses the proc filesystem provided by Linux. On the other hand, Linux also provides Netlink to communicate between kernel and user space. In this project inter-EE communication via Netlink should be implemented.

2 Assignment

2.1 Objectives

The following objectives are expected to be met in this semester thesis.

- Integrate Netlink in the Chameleon active node. It must be possible to use Netlink as a communication channel between Click EE and Chameleon Java EE.

- An active service must be defined and implemented executing components in either EE and communicating via Netlink. Additional communication via the proc filesystem is encouraged.

- Service descriptors for the implemented active service must be provided.

- Designing the service in a way that allows alternative implementations on the Chameleon node is encouraged.

2.2 Tasks

- Get familiar with active networks and service deployment concepts. Read and understand the assigned papers, which deal with this topic [5, 6, 2].

- Get familiar with Chameleon [1] and Click router [4].

- Install the Chameleon router on your PC [3].

- Get familiar with Linux Netlink.

- Implement Netlink support in Chameleon. Provide the necessary adapters for both EEs.

- Define an active service, the components of which run in both, Click EE and Chameleon Java EE. Communication between components in different EEs is done via Netlink (and possibly the proc filesystem).

- Implement the service and provide appropriate service descriptors.

- Implement a demonstration of the service. Scripts to carry out this demonstration must be provided.

- Document your work in a detailed and comprehensive way. We suggest you to continually update your documentation. New concepts and investigated variants must be described. Decisions for a particular variant must be justified.

3 Deliverables and Organisation

- If possible, student and advisor meet on a weekly basis to discuss progress of work and next steps. If problems/questions arise that can not be solved independently, the student may contact the advisor anytime.

- At the end of the third week, a detailed time schedule of the semester thesis must be given and discussed with the advisor.

- At half time of the semester thesis, a short discussion of 15 minutes with the professor and the advisor will take place. The student has to talk about the major aspects of the ongoing work. At this point, the student should already have a preliminary version of the table of contents of the final report. This preliminary version should be brought along to the short discussion.

- At the end of the semester thesis, a presentation of 15 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work. Furthermore, it should include a small demo of the project.

- The final report may be written in English or German. It must contain a summary written in English or German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Related work must be correctly referenced. See http://www.tik.ee.ethz.ch/~flury/tips.html for more tips. Three copies of the final report must be delivered to TIK.

- Documentation and software must be delivered on a CDROM

Anhang C: Aufgabenstellung

Literature

[1] Matthias Bossardt, Roman Hoog Antink, Andreas Moser, and Bernhard Plattner. Chameleon: Realizing automatic service composition for extensible active routers. In Proceedings of the Fifth Annual International Working Conference on Active Networks IWAN, number 2546 in Lecture Notes in Computer Science, Kyoto, Japan, December 2003.

[2] Matthias Bossardt, Lukas Ruf, Rolf Stadler, and Bernhard Plattner. A service deployment architecture for heterogeneous active network nodes. In IFIP International Conference on Intelligence in Networks (SmartNet), Saariselka, Finnland, April 2002. Kluwer Academic Publishers.

[3] R. Hoog Antink and A. Moser. Service composition for active networks. Master's thesis, ETH Zurich, Switzerland, 2003.

[4] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. ACM Transactions on Computer Systems, 18(3):263-297, August 2000.

[5] Konstantinos Psounis. Active networks, applications, safety, security, and architectures. IEEE Communication Survey Magazine, 2(1), 1999.

[6] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. IEEE Communications Magazine, 35(1): 80-86, January 1997.

Zürich, den 27.10.2003

Anhang D

Zeitplan

ZEITPLAN SEMESTERARBEIT

Woche / Thema	1 3.-9.11.	2 10.-16.11.	3 17.-23.11.	4 24.-30.11.	5 1.-7.12.	6 8.-14.12.	7 15.-21.12.	8 22.-28.12.	9 29.-4.1.	10 5.-11.1.	11 12.-18.1.	12 19.-25.1.	13 26.-1.2.	14 2.-8.2.	15 9.-13.2.
								Absenz von Betreuer							
Einarbeiten (Active Networks, Netlink, Service Deployment)															
Netlink					M1										
Service Design/Spezifikation					M2										
Service Implementation															
Service Deployment													M3		
XML-Beschreibungen															
Tests/Evaluation															
Dokumentation															
Demo/Präsentation														M4	

Absenz von Betreuer

Legende: M1: Netlink Kommunikation O.K.
M2: Service Szenario steht
M3: Service kann deployed werden
M4: Präsentation am 2.2.2004

Index