Projektarbeit
Active Networking mit ANTS

Marc Brogle
RVS-Gruppe
IAM Universität Bern

25. Mai 2000

Zusammenfassung

Dieses Informatikprojekt umfasst praktische Versuche in Active Networking mit dem ANTS-Toolkit (ANTS = Active Node Transfer System). Es wird ein Beispiel aus dem Bereich Quality Services (abhörsichere Verbindungen mit Hilfe von Paket-Splitting) betrachtet.

Das Projekt wurde hauptsächlich im Zeitraum vom Oktober 1999 bis Mai 2000 realisiert.

Betreuer des Projekts waren Florian Baumgartner (Idee) und Manuel Günter (Anforderungsdefinition, Umsetzung und Abnahme).

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung in Active Networking

1.1 Was ist Active Networking?

Mit Active Networking bezeichnet man eine neue Idee in der Netzwerkwelt, welche darauf beruht, dass Router nicht nur passive Pakete weiterleiten, sondern dass diese Pakete, welche verschiedene Router passieren, in diesen irgendwelche Aktionen ausführen können.
Die Liste der möglichen Aktionen, welche Pakete durchführen können ist eigentlich nur durch den Funktionsumfang der dabei verwendeten Programmiersprache beschränkt. Es folgt eine kurze Liste mit möglichen Ideen und Ansätzen:

- Verwerfen des Pakets (dies entscheidet das jeweilige Paket selbst, nicht der Router!)
- Verdoppelung des Pakets (oder gar x-fache Vervielfältigung), z.B. für Multicast, welches von den Paketen bestimmt wird
- Umwandeln des Pakets in ein anderes Format
- Umwandeln des Pakets in kleinere Teilpakete
- Änderung der Verschlüsselung, z.B. von 56 bit auf 128 bit, je nach dem, was auf der bevorliegenden Route für Schlüsselalgorithmen resp. Verschlüsselungsstärken erlaubt sind
- Zwischenspeichern des Pakets in einem Router

Es gibt verschiedene Möglichkeiten diese Ideen zu realisieren und in letzter Zeit sind diverse verschiedene Ansätze und Beispiel-Frameworks erstellt worden. Wir beschäftigen uns in dieser Projektarbeit mit dem ANTS-Toolkit, welches im folgenden Kapitel genauer beschrieben wird.

1.2 Was ist ANTS?

ANTS (Active Node Transfer System) ist ein Toolkit basierend auf Java und Tcl/Tk um Experimente mit Active Networks durchzuführen. Der Benutzer des Toolkits kann damit neue Knoten starten, die in einem Active Network teilnehmen und neue Protokoll-Modelle definieren, welche das Weiterleiten der Pakete in diesen Knoten spezifisch beeinflussen. ANTS wurde von der SDS Group[1] am MIT Laboratory for Computer Science[2] entwickelt. Einen kurzen Überblick über die Entstehung und die Möglichkeiten von ANTS findet man in [WLG98]. Ferner ist [CSAA98], [WGT98], [WGT99] und [Wet97] zur weiteren Vertiefung über Active Networking mit ANTS empfehlenswert.

[1] www.sds.lcs.mit.edu
[2] www.lcs.mit.edu

1.3 Terminologie von ANTS

Im folgenden werden erläutert, wie verschiedene Begriffe aus der Netzwerkwelt in ANTS verwendet werden und was diese bedeuten:

- Capsule: Pakete, welche durch ein ANTS-Netzwerk gesandt werden, werden Capsules genannt. Diese Capsules können sowohl Nutzdaten transportieren, als auch Algorithmen und Methoden beinhalten, welche an bestimmten Nodes (siehe weiter unten) ausgeführt werden können.
- Protocol: Verschiedene zusammengehörende Capsule-Typen werden in einem Protocol zusammengefasst. Ein Node (siehe weiter unten) registriert ein oder mehrere Protocols um diese Verarbeiten zu können.
- Node: Ein Node ist ein Netzwerk-Knoten, der wie ein Router Pakete auf dessen Wunsch weiterleitet.
- Nodecache: Jeder Node besitzt einen Nodecache, in welchem Pakete beliebige Objekte, also auch sich selbst, zwischenspeichern können. Die Objekte werden mit einem Hash-Key versehen, welcher auch wieder ein bliebiges Objekt sein kann. Nur Pakete vom gleichen Protokoll können auf Objekte im Cache, welche von anderen Paketen desselben Protokolls abgespeichert wurden zugreifen (dies ist mit Umwegen, jedoch durch Sicherheitsmechanismen geschützt, auch über die Protokollgrenzen hinaus möglich). Die abgespeicherten Objekte in einem Nodecache besitzen eine bestimmte Lebensdauer, welche von den Paketen selbst bestimmt werden kann. Siehe dazu auch Abbildung 1 auf Seite 7.
- Application: Applications (eine oder mehrere) werden an einen Node angehängt. Diese Applications können z.B. ein graphisches Frontend bereitstellen, Daten, welche der Node empfängt in Dateien abspeichern, Manipulation an den eintreffenden Paketen vornehmen, etc.

Alles in allem kann mit den oben erwähnten Objekten bereits eine Beispielapplikation aufgesetzt werden. Im Kapitel 2 wird der Aufbau und die Verwendung einer einfachen Ping-Applikation dargestellt.

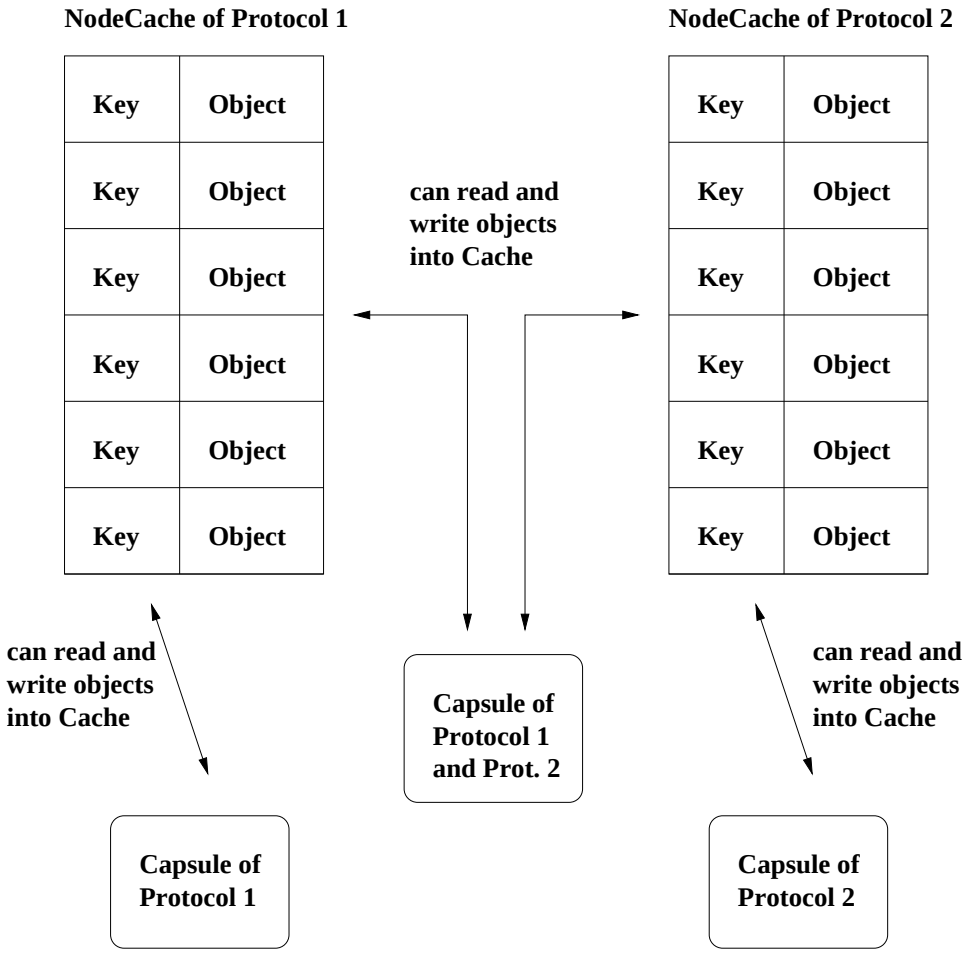

Abbildung 1: Aufbau des NodeChaches in einem einzelnen Node

2 Einführung in das Arbeiten mit ANTS

2.1 Systemanforderungen

ANTS setzt ein lauffähiges Linux-System voraus, auf welchem mindestens Java 1.02 oder neuer und Tcl/Tk installiert ist. An die Hardware werden keine speziellen Bedingungen gestellt, jedoch sollte, da Java verwendet wird, vorzugsweise eine rechenstarke CPU mit genügend Hauptspeicher zur Verfügung stehen.

2.2 Installation

Das ANTS-Paket kann von der offiziellen ANTS-Webseite[3] heruntergeladen werden und ist in einer GZIP-komprimierten TAR-Datei gepackt (Version 1.2: `ants-release-11-2-97.tar.gz`). `Mit tar -xzvf` *`Dateiname`* wird im aktuellen Verzeichnis ein Unterverzeichnis mit den entpackten ANTS-Programmdateien erstellt (Version 1.2: `ants-1.2`).
Weiterhin sollte die Umgebungsvariable `CLASSPATH`, welche von Java verwendet wird, mit dem Pfad des ANTS-Toolkits und dem des aktuellen Verzeichnis erweitert werden (hier als Beispiel für die Version 1.2 im Homeverzeichnis des Autors):

```
export CLASSPATH='$CLASSPATH:.:/home/brogle/ants-1.2'
```

Ferner sollte die Umgebungsvariable `PATH` um das Unterverzeichnis `/runs` in dem ANTS-Toolkit-Verzeichnis erweitert werden (hier als Beispiel für die Version 1.2 im Homeverzeichnis des Autors):

```
export PATH=/home/brogle/ants-1.2/runs.:$PATH
```

Bevor man nun das Toolkit verwenden kann, muss es noch komplett kompiliert werden. Dies geschieht durch Eingabe von `make all` im ANTS-Toolkit-Verzeichnis.

2.3 Aufbau der Verzeichnisse des ANTS-Toolkit

ants enthält die Implementierung des aktiven Knoten-Programms und die abstrakten Schlüsselklassen

apps enthält die (auch selbstgeschriebenen) Applikationen mit ihren Capsules, Protocols, Nodes, etc. und Beispielprogrammen

auction enthält das umfangreichere Beispielprogramm für eine Online-Auktion

docs enthält verschiedene Dokumente u. a. [WGT98] und [Wet97], und die von `javadoc` beim Kompilieren des Pakets automatisch generierte API-Referenz und Dokumentation

[3] www.sds.lcs.mit.edu/activeware/ants/

runs enthält alle Konfigurationsdateien, die Startdateien, Logdateien und die Standardbeispiele, welche dem ANTS-Toolkit mitgeliefert werden

utils enthält ausgesuchte und allgemeine Hilfsprogramme

2.4 Aufbau einer Beispielkonfiguration (PingRVS)

Das Verhalten des ANTS-Toolkits wird durch eine Konfigurationsdatei (enthält die Definition der einzelnen Knoten und Ihre Vernetzung untereinander), einer (durch ein Hilfsprogramm[4]) automatisch erstellten weiteren Konfigurationsdatei (welche alle möglichen Netwerkpfade enthält) und schliesslich die entsprechenden selbst erstellten Protokolle (in Java geschrieben).
Im folgenden betrachten wir nun eine Beispielanwendung, welche einen einfachen PING-Befehl simuliert. Sie ist konfiguriert für den Linux-Pool der RVS Gruppe an der Universität Bern und verwendet die beiden Rechner *sam* und *coyote*.

2.4.1 Aufbau der Konfigurationsdatei

```
# coyote (server)
node 18.31.12.11 -routes PingRVS.routes
channel 18.31.12.11 sam:8011 -log 0
application 18.31.12.11 apps.DataServerApplication -target 18.31.12.13

# coyote (router)
node 18.31.12.12 -routes PingRVS.routes
channel 18.31.12.12 sam:8012 -log 0

# sam (client)
node 18.31.12.13 -routes PingRVS.routes
application 18.31.12.13 apps.DataClientApplication
channel 18.31.12.13 coyote:8013

connect 18.31.12.11 18.31.12.12
connect 18.31.12.12 18.31.12.13
```

Listing: PingRVS.config

Im folgenden ein paar Erläuterungen zu den einzelnen Abschnitten der Konfigurationsdatei (PingRVS.conf):

- Einzeilige Kommentare werden mit # eingeleitet.

[4] makeroutes

- Ein neuer Knoten wird mit `node` *`Adresse`* `-routes` *`Dateiname`* erzeugt. Als erster Parameter wird eine Active-Network-Adresse dem Knoten zugewiesen (im IP-Format) und mit dem Parameter `-routes` wird dem Knoten mitgeteilt, welches die Datei mit den Netzwerkpfadinformationen ist.

- Ein neuer Kanal (muss jedem Knoten zugeteilt werden!) wird durch den Befehl `channel` *`Adresse computername:portnummer`* erzeugt. Mit dem zusätzlichem Parameter `-log` *`Zahl`* wird eine Logdatei mit Aufstartinformationen des Knotens erzeugt.

- Den Knoten werden die entsprechenden Protokolle mit dem Befehl `application apps.`*`Protokollname [Optionen]`* respektive `manager apps.`*`Protokollname [Optionen]`* zugewiesen.

- Am Ende der Datei müssen noch die einzelnen Verbindungen zwischen zwei direkt verbundenen Knoten mit dem Befehl `connect` *`Adresse1 Adresse2`* konfiguriert werden.

In der Beispieldatei wird also auf dem Rechner *coyote* je ein Server-Knoten und ein Router-Knoten erzeugt, auf dem Rechner *sam* wird der Client-Knoten eingerichtet. Das virtuelle Netzwerk besteht aus 2 Verbindungen. Die erste Verbindung ist zwischen Client und Router (*sam* $\rightarrow$ *coyote*). Die zweite Verbindung besteht zwischen Router und Server (*coyote* $\rightarrow$ *coyote*). Siehe dazu auch die Abbildung.

2.4.2 Aufbau der Netzwerkpfaddatei (automatisch generiert)

```
#
# shortest routes, automatically generated
# by ./makeroutes on 02/28/00
#

# source destination next    addr

18.31.12.11 18.31.12.12 18.31.12.12 sam:8012
18.31.12.11 18.31.12.13 18.31.12.12 sam:8012
18.31.12.12 18.31.12.11 18.31.12.11 sam:8011
18.31.12.12 18.31.12.13 18.31.12.13 coyote:8013
18.31.12.13 18.31.12.11 18.31.12.12 sam:8012
18.31.12.13 18.31.12.12 18.31.12.12 sam:8012
```

Listing: PingRVS.routes

Die Netzwerkpfaddatei enthält alle möglichen Verbindungen der Knoten untereinander. Sie wird mit dem Tk/Tcl-Skript `makeroutes` automatisch anhand der `connect`-Anweisungen in der Konfigurationsdatei generiert. Als Beispiel dient die Datei PingRVS.routes für die Datei PingRVS.conf. Für einfache Netzwerktopologien ist die Erstellung trivial auch von Hand möglich.

2.4.3 Aufbau der Startdateien für die einzelnen Knoten

Im Folgenden die Startdateien für die 2 Rechner *coyote* (PingRVScoyote.start) und *sam* (PingRVSsam.start). Die entsprechenden Dateien werden mit `csh PingRVScoyote.start` (auf dem Rechner *coyote*) und `csh PingRVSsam.start` (auf dem Rechner *sam*) gestartet.

```
#!/bin/csh
java ants.ConfigurationManager PingRVS.config 18.31.12.12 >& 18.31.12.12.log &
java ants.ConfigurationManager PingRVS.config 18.31.12.11
kill %?18.31.12.12
```

Listing: PingRVScoyote.start

```
#!/bin/csh
java ants.ConfigurationManager PingRVS.config 18.31.12.13
```

Listing: PingRVSsam.start

2.4.4 Aufbau der JAVA-Klassen

Im Folgenden die Sourcecodes der 2 Javaklassen DataClientApplication und DataServerApplication, welche das Verhalten der aktiven Knoten in der PingRVS Beispielkonfiguration bestimmen.

```
package apps;

import java.awt.*;

import ants.*;
import utils.*;

public class DataClientApplication extends Application
```

```
{
  final public static String[] defaults = { "-delay", "0" };
  int target, delay, rTotCount = 0, thruInterval = 100;
  final int oneLess = 99;
  long beginTime;

  synchronized public void receive(Capsule cap) {
    super.receive(cap);

    if (cap instanceof DataCapsule) {
      switch ((rTotCount % thruInterval)) {
        //case 0:
        //beginTime = System.currentTimeMillis();
        //break;
      case oneLess:
          double diff =
  (double) (System.currentTimeMillis() - beginTime) / 1000.0;
        double thru = ((double) thruInterval) / diff;
        beginTime = System.currentTimeMillis();
        System.out.println("Received the " + (rTotCount + 1) + "th capsule");
        System.out.println("Throughput for last " + thruInterval +
                    " capsules is " + thru + " caps/sec");
        System.out.flush();
        break;
      }

      rTotCount++;
    }
  }

  public void setArgs(KeyArgs k)
    throws Exception
  {
    k.merge(defaults);

    for (int i = 0; i < k.length(); i++) {
      if (k.key(i).equals("-target")) {
        target = NodeAddress.fromString(k.arg(i));
        k.strike(i);
      } else if (k.key(i).equals("-delay")) {
        delay = Integer.parseInt(k.arg(i));
         k.strike(i);
      }
    }

    super.setArgs(k);
  }

  public void start()
```

```
    throws Exception
  {
    thisNode().register(new NullDataProtocol());
  }

  public DataClientApplication()
    throws Exception
  {
    super();
    beginTime = System.currentTimeMillis();
  }
}
```

Listing: DataClientApplication.java

```
package apps;

import java.awt.*;

import ants.*;
import utils.*;

public class DataServerApplication extends Application implements Runnable
{
  final public static String[] defaults = { "-delay", "0" };
  int target, delay;
  Button sender;
  Label output;
  TextField iterations, interval;

  synchronized public void receive(Capsule cap) {
    super.receive(cap);
    output.setText("What the hell?\nI just received a packet.");
  }

  public boolean handleEvent(Event evt) {
    if (evt.id == Event.ACTION_EVENT && evt.target == sender) {
      new Thread(this).start();
      return true;
    } else
      return super.handleEvent(evt);
  }

  public void run()
  {
    int iter = Integer.parseInt(iterations.getText());
    double interv = Double.valueOf(interval.getText()).doubleValue();
```

```
    long itvMillis = (long) Math.floor(interv), beg, diff;
    int itvNanos = (int) Math.round((interv - Math.floor(interv))*1000000);

    long begTime, endTime;

    for (int i=0; i < iter; i++) {
      beg = System.currentTimeMillis();
      DataCapsule c = new DataCapsule(port, port, target, new ByteArray(256));
      send(c);
      if (i%100==99)
        System.out.println("Have sent " + (i+1) + " capsules");
      diff = System.currentTimeMillis() - beg;
      if (itvMillis >= diff) {
        try {
  Thread.sleep(itvMillis - diff, itvNanos);
        }
        catch (InterruptedException e) {
          e.printStackTrace(); }
      }
    }
  }

  public void setArgs(KeyArgs k)
    throws Exception
  {
    k.merge(defaults);

    for (int i = 0; i < k.length(); i++) {
      if (k.key(i).equals("-target")) {
        target = NodeAddress.fromString(k.arg(i));
        k.strike(i);
      } else if (k.key(i).equals("-delay")) {
        delay = Integer.parseInt(k.arg(i));
        k.strike(i);
      }
    }

    super.setArgs(k);
  }

  public void start()
    throws Exception
  {
    thisNode().register(new DataProtocol());

    resize(400, 200);

    // 2 rows by 2 columns with 2 pixel point spaces in between them
    setLayout(new BorderLayout());
```

```
    sender = new Button("send " + NodeAddress.toString(target));

    output = new Label("Nothing to report", Label.CENTER);

    Panel iterDisplay = new Panel();
    iterDisplay.setLayout(new BorderLayout());
    iterDisplay.add("Center",
    new Label("Num Iterations", Label.CENTER));
    iterations = new TextField("0");
    iterations.setEditable(true);
    iterDisplay.add("South", iterations);

    Panel intervalDisplay = new Panel();
    intervalDisplay.setLayout(new BorderLayout());
    intervalDisplay.add("Center",
        new Label("Send interval (in ms)", Label.CENTER));
    interval = new TextField("0");
    interval.setEditable(true);
    intervalDisplay.add("South", interval);

    Panel topPanel = new Panel(), botPanel = new Panel();
    topPanel.setLayout(new GridLayout(1, 2, 1, 1));
    botPanel.setLayout(new GridLayout(1, 2, 1, 1));

    topPanel.add(sender);
    topPanel.add(output);
    botPanel.add(iterDisplay);
    botPanel.add(intervalDisplay);

    add("South", botPanel);
    add("Center", topPanel);
    show();
  }

  public DataServerApplication()
    throws Exception
  {
    super();
  }
}
```

Listing: DataServerApplication.java

3 Die SplitPad-Applikation

Dieser Abschnitt beschreibt die SplitPad-Applikation, wie sie implementiert wurde, die allgemeinen Überlegungen zur Gewährleistung der Datensicherheit und auch weiterführende Überlegungen und mögliche Erweiterungen der hier vorgestellten Lösung. Die ganze Applikation wurde mit dem ANTS Framework realisiert und somit komplett in Java geschrieben. Siehe dazu auch Kapitel 2 in diesem Dokument und die entsprechende Literatur, welche im Literaturverzeichnis erwähnt wird.

3.1 Die Idee hinter SplitPad

Die Idee hinter SplitPad lässt sich wie folgt beschreiben:

> Sensible Daten sollen durch ein Netzwerk gesendet werden, jedoch vertraut man einigen involvierten Knotenpunkten (dies können Router, PCs und Ähnliches sein) nicht. Daher werden die Daten in 2 disjunkte Teilpakete aufgeteilt und über separate Pfade gesendet. Die eine Hälfte der Daten, also das erste zugehörige Teilpaket enthält den Verschlüsselungscode, während die andere Hälfte, das zweite zugehörige Teilpaket die verschlüsselten Daten enthält. Jedes Paket wird mit einem neuen Schlüssel verschlüsselt.

Auf weitere Details zu der SplitPad-Idee wird im Verlaufe dieses Kapitels noch genauer eingegangen. Zuerst soll jedoch kurz anhand einiger Grafiken ein besseres Verständnis über ein mögliches Szenarios geschaffen werden.
Die Abbildung 2 auf Seite 17 zeigt ein mögliches Netzwerk an welchem Sender und Empfänger angeschlossen sind. Die Netzwerkknoten sind mannigfaltig untereinander vernetzt. In diesem Netzwerk existieren jedoch auch Knoten von Parteien, welchen der Sender resp. Empfänger nicht trauen. Es muss also ein Weg durch das Netzwerk gewählt werden, welcher die erwähnten Knoten geschickt einbezieht. Siehe dazu Abbildung 3 auf Seite 18. In der Abbildung 4 auf Seite 19 vereinfachen die Darstellung und konzentrieren uns auf die wesentlichen Knoten. Dieser Aufbau wird auch in den in diesem Dokument beschriebenen Testszenarios verwendet. Es gilt nun kurz vor den kritischen Knoten die SplitKnoten aufzusetzen und ein Teil der Information über den einen kritischen Weg und den anderen Teil der Information über den zweiten kritischen Weg zu lenken. Nachdem diese Informationen die beiden kritischen Wege durchquert haben, kann man die Information wieder zusammensetzen und entsprechend zum Ziel weiterleiten.
Die Information wird mit Hilfe von Zufallszahlen, welche unter anderem aus dem Delay resp. Jitter von den einzelnen gesendeten Pakete generiert werden, mit einer einfachen (aber schnellen) XOR-Verknüpfung codiert. Dabei wird der Codierungsschlüssel über den einen Teilweg gesendet, während die

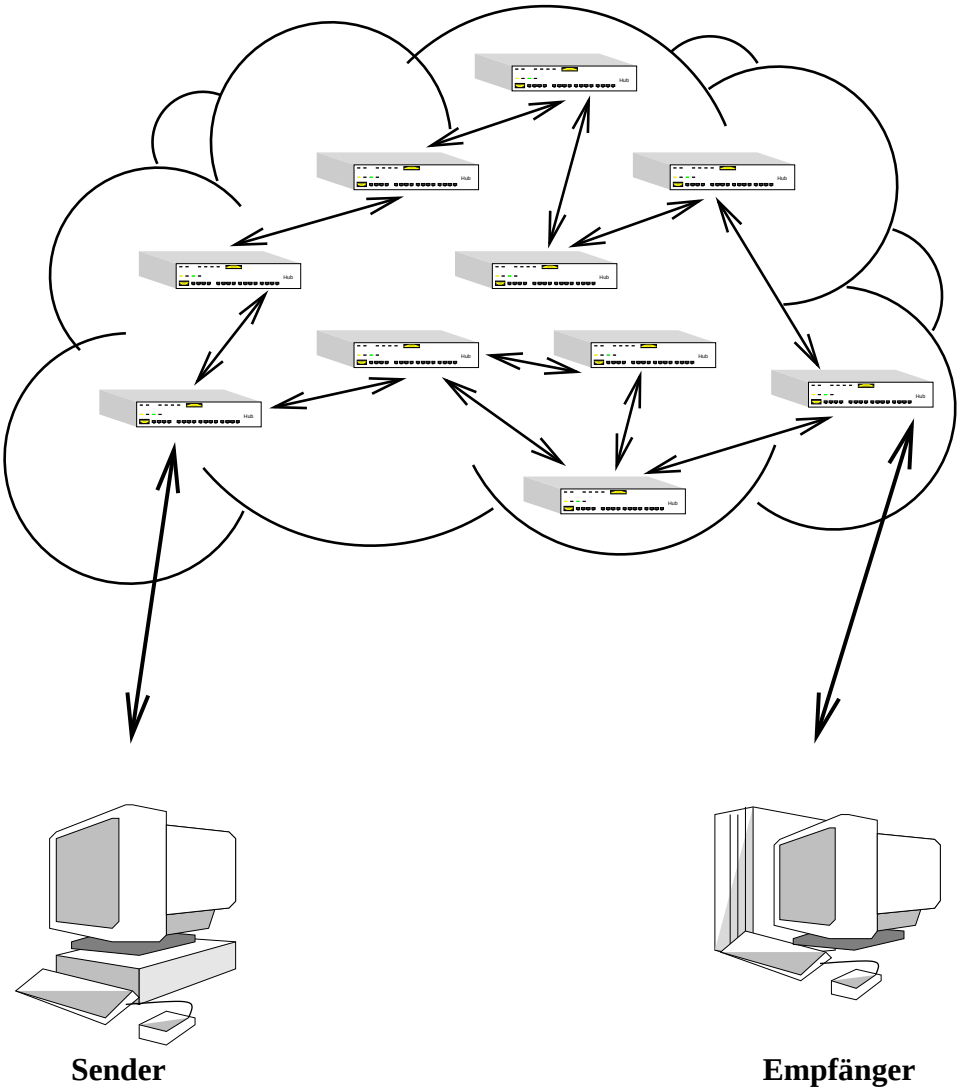

Abbildung 2: Beispielaufbau eines komplexen Netzwerks

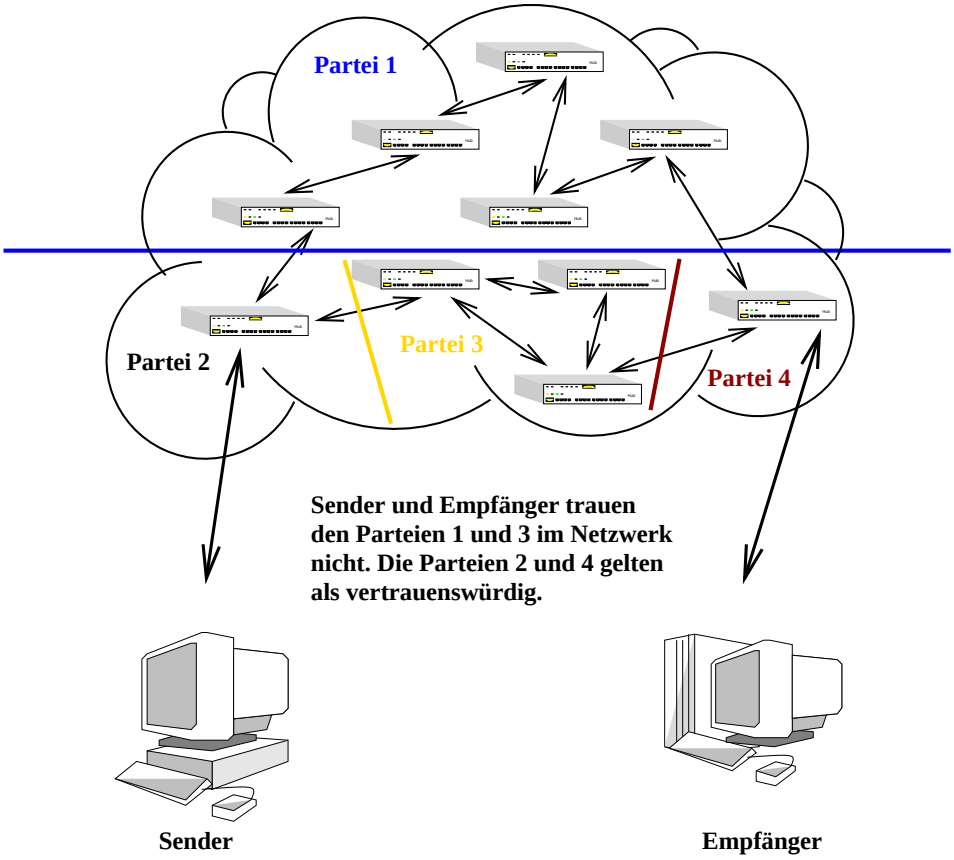

Abbildung 3: Beispielaufbau mit ausgewähltem Pfad

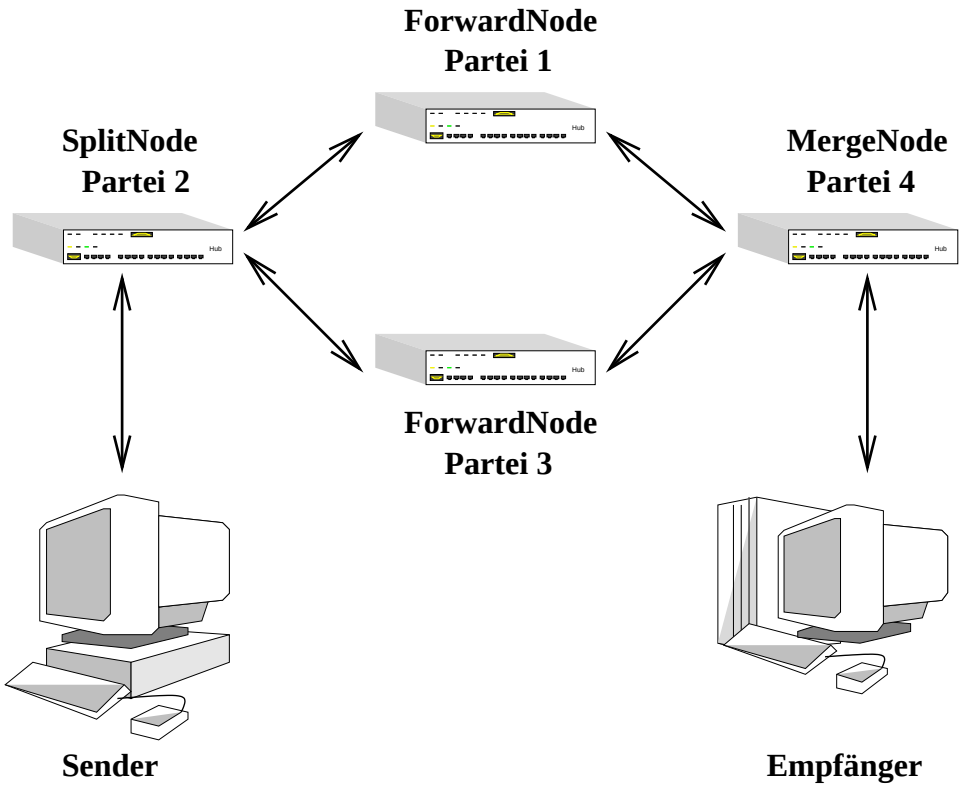

Abbildung 4: Vereinfachte Darstellung des Beispielaufbaus

codierten Daten über den anderen Teilweg gesendet werden. Auf die genaue Generierung der Zufallszahlen (resp. des Random-Seeds, welches aus diesen Zufallszahlen gewonnen wird) mit Hilfe des oben beschriebenen Verfahrens, resp. die Analyse der Qualität dieses erzeugten Random-Seeds, wird genauer in [GBB] eingegangen.
Für die Implementation der verschiedenen Tests wurde immer das oben gezeigte einfache Modell eines Netzwerks verwendet. Knoten 1 ist der Sendeknoten, der 2. Knoten (der Split-Knoten) teilt die Informationen auf und leitet diese geteilten und verschlüsselten Daten an die Knoten 3 resp. 4 weiter, welche wiederum diese an den Knoten Nummer 5 weiterleiten, welcher die Daten wieder decodiert und zusammenfügt (der Merge-Knoten). Schliesslich wird diese komplette Information wieder an den Knoten 6 weitergeleitet, welcher den Empfänger darstellt.

3.2 Die 2 Varianten der SplitPad-Applikation

Es wurden 2 Applikationen zur Realisierung der SplitPad-Idee geschrieben:

- die erste Variante erlaubt das Senden kurzer Nachrichten, wobei die entsprecheden Split- und Merge-Knoten und diverse weitere Konfigurationsmöglichkeiten interaktiv aufgesetzt und angepasst werden können, dabei wird ausgiebig Debug-Ausgabe produziert. Diese Variante eignet sich um sich ein Verständnis der SplitPad-Idee aufzubauen.

- die zweite Variante erlaubt das Senden einer Datei, welche via komfortablem Dialog ausgesucht werden kann. Weiterhin kann angegeben werden wie oft die Datei gesendet werden soll. Es können keine Konfigurationseinstellungen vorgenommen werden, jedoch wird Debug-Information wie z.B. aktuelles "Alter" des Pakets in Millisekunden resp. Aufbau des Random-Seeds in ein Log-File geschrieben. Diese Variante eignet sich um Performancemessungen und statistische Tests für die Überprüfung der Qualität der Zufallszahlen-Erzeugung durchzuführen.

Die erste Variante lässt sich durch Aufrufen des Startskripts `sp.start` in dem Unterverzeichnis `ants-1.2/runs` ausführen. Die 2. Variante im selben Verzeichnis durch Aufruf von `spf.start`. Gegebenenfalls müssen die entsprechenden Konfigurationsdateien `sp.conf` und `spf.conf` angepasst und durch Aufruf von `make` neu übersetzt werden.

3.3 Übersicht der Klassen für die SplitPad-Applikationen

Folgende Klassen werden für die SplitPad Applikation gebraucht:

- **SPProtocol**

- **SPCapsuleNormal**

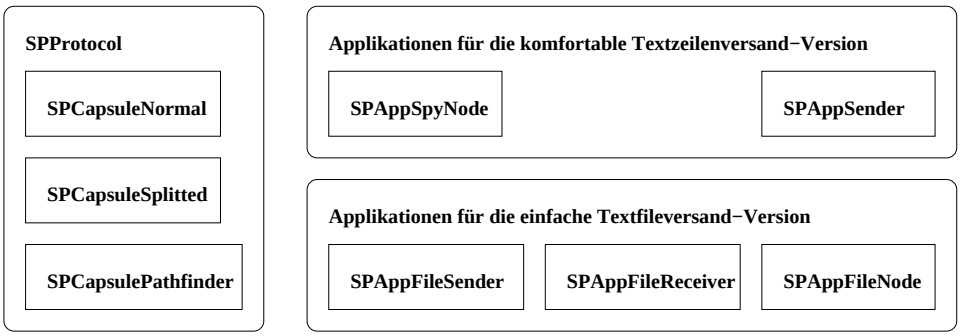

Abbildung 5: Überblick und Zusammenhänge der einzelnen Klassen

- **SPCapsuleSplitted**
- **SPCapsulePathfinder**
- **SPAppSender**
- **SPAppSpyNode**
- **SPAppFileReceiver**
- **SPAppFileSender**
- **SPAppFileNode**

Einen Überblick über die Zusammenhänge der einzelnen Klassen sehen sie in der Abbildung 5 auf Seite 21.

3.4 Die Klassen für SplitPad im Detail

3.4.1 SPProtocol

Diese Klasse definiert die involvierten Pakettypen zusammen in einem Protocol. Jeder Node registriert eine oder mehrere Protocols. Diese Zusammenfassung in einem Protocol ist nötig, damit Pakete (in ANTS Capsules genannt) in einem Node auf zwischengespeicherte Werte zugreifen zu können, welche andere Capsules des gleichen Protocols abgespeichert haben. Dies gestattet eine gewisse Sicherheit in den Knoten, welche auf andere Protokolle resp. Capsules verarbeiten. Es können also nur innerhalb eines Protocols Capsules auf diese zwischengespeicherten Daten in den Knoten zugreifen.

3.4.2 SPCapsuleNormal

Die Capsule dieses Typs ist im ungesplitteten Zustand und trägt die (unverschlüsselten) Nutzdaten. An jedem Knoten, den es besucht, wird nach einem Eintrag in Cache gesucht, welcher der Capsule mitteilt, ob sie sich an

einem normalen Knoten (nur Forwarding) oder an einem Splitknoten (die Capsule muss sich hier teilen) befindet. Im Falle eines normalen Knotens wird die Capsule einfach Richtung Ziel weitergeleitet. Ist sie jedoch an einem Splitknoten, so sucht sie im Cache des Knotens nach den 2 Zieladressen für die beiden Hälften, holt das Random-Seed aus dem Knoten (resp. generiert ein neues, falls das im Cache vorhandene Randomseed eine gewisse Dauer und Anzahl von Nutzungen bereits hinter sich hat oder noch gar kein Randomseed existiert).

3.4.3 SPCapsuleSplitted

Eine SPCapsuleSplitted ist die gesendete Information im gesplitteten Zustand. An jedem Konten an dem sie vorbeigelangt überprüft sie, ob sie ihren Zielknoten, um sich wieder mit ihrer entsprechenden anderen Hälfte zusammenzufügen, schon erreicht hat. Ist dies der Fall, so überprüft sie, ob im Cache des Knotens ihr Gegenstück bereits gespeichert ist (Identifikation durch 2 in jeder Capsule gespeicherte Variablen, welche die Zeit der Entstehung der Capsule und die Sequenznummer enthalten). Ist dies der Fall, so holt sie die andere Hälfte aus den Cache und diese beiden fügen sich wieder zu einer SPCapsuleNormal zusammen, welche weiter Richtung Zielknoten gerouted wird. Ist sie jedoch alleine an dem Knoten, das heisst ihr Gegenstück ist noch nicht eingetroffen, so speichert sie sich selbst in dem Cache ab (mit einer bestimmten Lebenserwartung).

3.4.4 SPCapsulePathfinder

Dieser Typ einer Capsule baut den Weg durch das Netzwerk auf. Einer SPCapsulePathfinder wird mitgeteilt, zu welchem Knoten sie sich routen lassen soll und was für eine Art Knoten sie dort für die anderen Capsules konfigurieren soll (also ob ein Split- oder Mergeknoten für die entsprechenden Capsules vorliegt. Die SPCapsulePathfinder kann zu einem beliebigen Knoten als ihr Ziel gerouted werden, aber dabei irgendwo auf dem Weg einen Knoten konfigurieren. Ihre Zieladresse ist nicht zwangsläufig der Knoten, welche sie konfigurieren muss. Dies ist aus dem Aspekt der Sicherheit von Vorteil, da bei einem beliebigen Knoten das Ziel für diese Capsule ausgelesen werden könnte, und man daraus möglicherweise auf einen möglichen Split- resp. Mergeknoten an dessen Ziel schliessen könnte.

3.4.5 SPAppSpyNode

Diese Klasse ist eine Applikation, welche an einen Knoten angehängt wird, der als Ziel für die erwähnten Capsules fungiert oder zur Überprüfung der Capsules, welche durch einen bestimmten Knoten (Split-, Mergeknoten oder ein normaler Knoten) gerouted werden. Es werden hier genaue Informationen über die entsprechende Capsule ausgegeben. Somit kann überprüft werden,

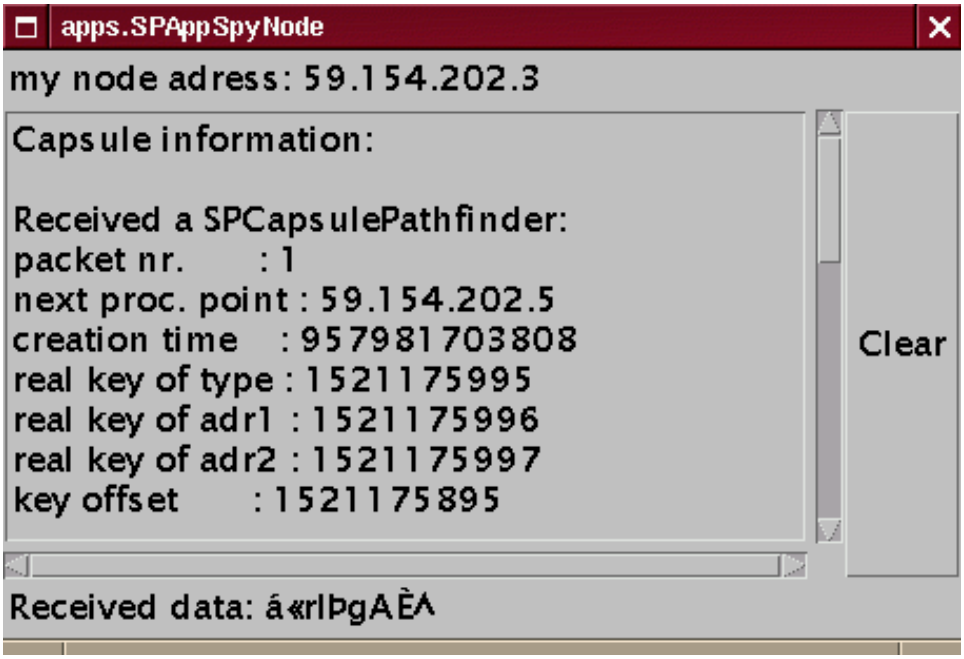

Abbildung 6: Frontend zu einem SPAppSpyNode Bsp. 1

ob sich das aufgesetzte Netzwerk nach den voreingestellten Konfigurationen verhält. Weiterhin wird eine umfangreiche Logdatei erzeugt, welche unter anderem auch die bisherige Lebenszeit einer Capsule enthält. Diese Informationen können zur Erstellung von statistischen Daten (Performance, echte Verteilung des Randomseeds, usw.) verwendet werden. Siehe dazu auch Abbildung 6 auf Seite 23 und Abbildung 7 auf Seite 24.

3.4.6 SPAppSender

Mit Hilfe der Klasse SPAppSender können die diversen Capsules gesendet werden. Einerseits erlaubt diese Applikation, die wie SPAppSpyNode an einen Knoten angehängt wird, die Konfiguration und das Senden von SPCapsulePathfinder-Capsules als auch das Absenden von Daten mit Hilfe einer SPCapsuleNormal. Diese Version zum Senden erlaubt es einem kurze Strings, also Textnachrichten durch das vorher aufgesetzte Netzwerk zu senden. Siehe dazu auch Abbildung 8 auf Seite 24.

3.4.7 SPAppFileReceiver

Die Klasse SPAppFileReceiver erlaubt das Empfangen einer Datei. Sie besitzt kein Frontend zum Benutzer sondern nimmt die empfangenen Daten stillschweigend entgegen und speichert diese, falls erwünscht, in einer Datei ab. Es wird eine einfache Logdatei erzeugt, welche die Lebensdauer emp-

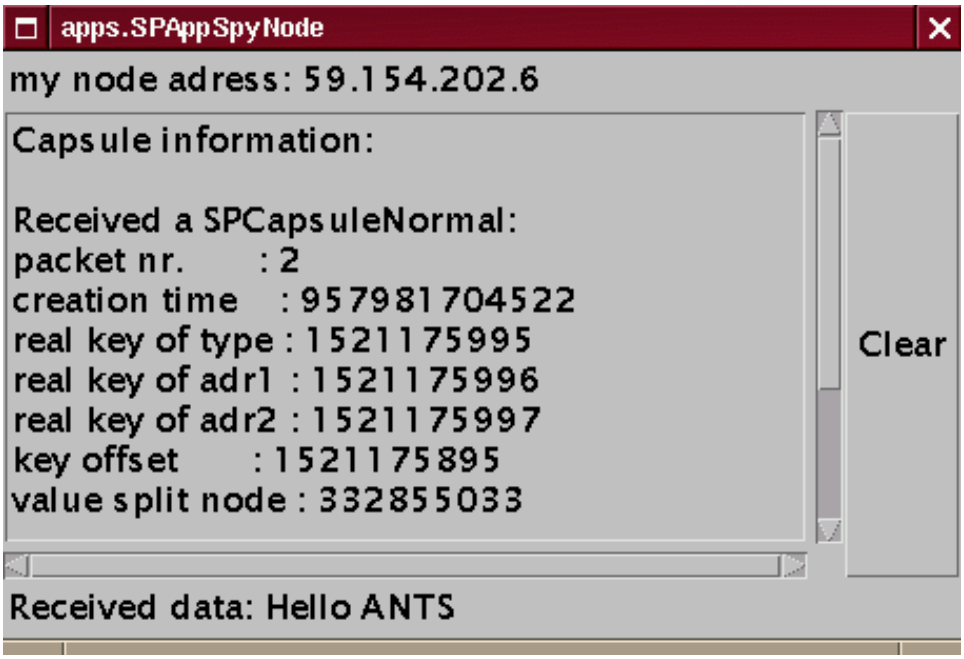

Abbildung 7: Frontend zu einem SPAppSpyNode Bsp. 2

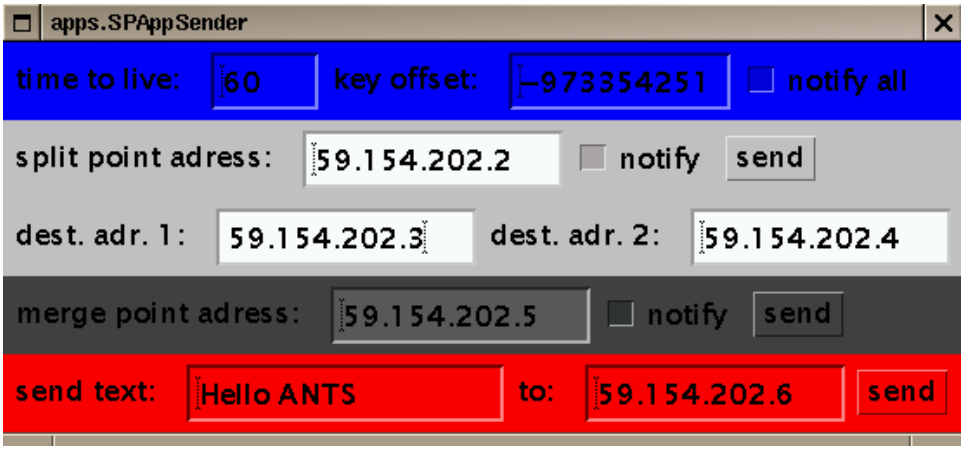

Abbildung 8: Frontend zu einem SPAppSpender

Abbildung 9: Frontend zu einem SPAppFileSender

fangener Capsules abspeichert. Die Konfiguration der einzelnen Parameter erfolgt ausschliesslich im Quellcode der Datei. Nach Anpassung der Konfiguration muss die Klasse neu kompiliert werden.

3.4.8 SPAppFileSender

Mit Hilfe der Applikationsklasse SPAppFileSender, welche auch wieder einem Knoten angehängt wird, lässt sich eine Datei 1 oder mehrmals an einen SPAppFileReceiver-Knoten senden. Die Konfiguration der Parameter erfolgt in dem Quelltext der Klasse (muss also bei Anpassung neu übersetzt werden). Der Benutzer hat jedoch auch ein Frontend zur Verfügung, welches ihm erlaubt die Datei komfortabel auszuwählen und die Anzahl der Wiederholungen des Sendevorgangs zu bestimmen. Siehe dazu auch Abbildung 9 auf Seite 25 und Abbildung 10 auf Seite 26.

3.4.9 SPAppFileNode

SPAppFileNode ist eine sehr einfach Klasse nach dem Muster der Klasse SPAppSpyNode. Sie erzeugt jedoch nur eine Logdatei, welche die Lebensdauer der einzelnen Pakete enthält. Sie besitzt auch kein Frontend zum User, um möglichst keinen direkten Einfluss auf die Performance zu nehmen.

3.5 Allgemeine Überlegungen zur Gewährleistung der Sicherheit

Es wurden diverse Massnahmen getroffen um die Sicherheit des kompletten SplitPad-Systems gegen verschiedene Abhörmöglichkeiten und Sabotageabsichten zu schützen. Im Folgenden werden die bereits implementierten Ideen erläutert:

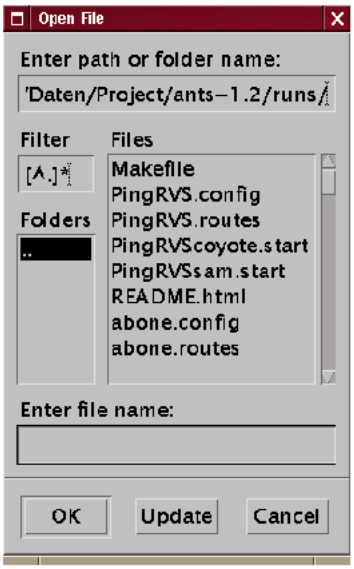

Abbildung 10: "File öffnen" Dialog eines SPAppFileSender

3.5.1 Zufällige Schlüssel-Werte für die Node-Cache-Einträge

Die Schlüsselwerte, welche die Verschiedenen Einträge in den Node-Caches identifizieren, werden zufällig erzeugt und haben nur eine bestimmte Lebensdauer. Diese Werte für die Schlüssel werden bei jedem Start des Sendeprogramms neu generiert. Zur Vereinfachung, wird einmalig ein Offset (zufälliger Integerwert) generiert, welcher zu den einzelnen Werten der Schlüssel hinzugezählt wird. Siehe dazu auch Abbildung 11 auf Seite 28.
Diese Werte umfassen die Schlüssel für:

- die Kennung für die Art eines Nodes (Split oder Merge): 100 + Offset
- die beiden Zieladressen für eine SplitNode: 101 + Offset resp. 102 + Offset

Im Endeffekt wird einer Capsule, welche die Daten enthält, nur noch der Offsetwert zur Identifizierung der Schlüsselwerte mitgegeben.

3.5.2 Zufällige Werte für die Werte welche einen Node konfigurieren

Neben den Schlüssel-Werten, werden auch verschiedene Inhalte für die Node-Cache Werte zufällig erzeugt.
Dies sind:

- der Wert, welcher einen Knoten als Split-Node definiert
- der Wert, welcher einen Knoten als Merge-Node definiert

Diese Werte werden auch beim Start des Sendeprogramms erzeugt, und gelten wie die Schlüssel-Werte für die ganze Dauer, in der das Programm läuft.

3.5.3 Zieladresse einer CapsulePathfinder ist nicht die Adresse des zu konfigurierenden Nodes

Wie bereits erwähnt, ist die Zieladresse einer SPCapsulePathfinder nicht unbedingt die Adresse des Knotens, welcher durch diese Capsule konfiguriert werden soll. Siehe dazu auch Kapitel 3.4.4.

3.5.4 Adressen der Split- und Merge-Nodes sind nicht in den Daten-Capsules gespeichert

Die Adressen der Knoten, welche eine SPCapsuleNormal dazu veranlassen sich zu teilen, resp. 2 SPCapsuleSplitted dazu veranlassen sich wieder zu einer SPCapsuleNormal zusammenzufügen, sind nicht in den entsprechenden Capsules selbst gespeichert, sondern nur die Schlüsselwerte der Einträge in den NodeCaches der einzelnen Nodes, welche den Typ des Nodes auszeichnen resp. die entsprechenden Zieladressen. Zusätzlich wird den Capsules auch

NodeCache eines SPAppSpyNode

Hash−Key	**beliebiges Objekt**
Key1 + Offset	**Kennung des NodeTypes**
Key2 + Offset	**ggf. Zieladresse 1 für SplitCap.**
Key3 + Offset	**ggf. Zieladresse 2 für SplitCap.**
CapsuleKey1	**komplette SPCapsuleSplitted**
.....	
CapsuleKeyN	**komplette SPCapsuleSplitted**

Offset (fix) = 2435

Key1 = 100
Key2 = 101
Key3 = 102

− Hash Key Einträge sind vom Typ Integer

− nur Cpasules welche im gleichen Protocol registriert sind können auf die NodeCache−Einträge zugreiffen

− CapsuleKeyX ist ein zusammengesetzter Eintrag vom Typ String in der Form "creationtime:sequencenumber", welche von gleichen gesplitteten Capsules gebraucht wird um ihr jeweiliges Gegenstück im Cache ausfindig zu machen

Abbildung 11: NodeCache-Schnappschuss eines SPAppSpyNode

noch die entsprechenden Werte für die Identifizierung der Inhalte an den erwähnten Stellen im NodeCache mitgegeben. Daraus folgt, dass man anhand einer SPCapsuleNormal resp. SPCapsuleSplitted nicht feststellen kann, welches die entsprechenden Split- und Merge-Knoten in einem Netzwerk sind.

3.5.5 Fazit über die eingesetzten Sicherheitsmassnahmen

Die hier gezeigten Massnahmen zur Sicherheit, stellen ein gutes Gleichgewicht zwischen Sicherheit und Performance dar. Sie können zwar noch im grossen Umfang erweitert werden (siehe dazu auch Kapitel 3.7), sind aber zum Aufzeigen der Möglichkeiten und zur Implementierung des bestehenden Prototypen völlig ausreichend.

3.6 Probleme der gewählten Implementation und deren Lösung

3.6.1 Probleme der bestehenden Implementation

In der gezeigten Implementierung werden nur die Split- und Merge-Knoten konfiguriert, resp. gekennzeichnet. Dies hat den Vorteil, dass relativ schnell auch grosse Netze mit diversen Split- und Sende-Nodes aufgesetzt werden können. Man muss sich nur Gedanken über das Forwarding an den Split- und Merge-Nodes machen, nicht aber die sonstige Struktur des Netzes, also "normale" Knoten, konfigurieren. Dies kann jedoch bei schlechter Planung oder unvollständigen Angaben über das Netzwerk möglicherweise zu Deadlocks und Zyklen führen. Es gibt im Grossen und Ganzen 3 mögliche Szenarien:

- Nehmen wir einmal an, man konfiguriert sich einen Weg, auf dem nach dem Splitten der Capsules, die entsprechenden Teilwege gar nicht mehr zusammen finden können. Somit werden nun nur Capsules SPCapsuleSplitted beim Ziel ankommen. Dies ist weniger tragisch, da die dazwischenliegenden Knoten, welchen man nicht vertraut auch nur verschlüsselte Informationen erhalten.

- Es wird ein Weg aufgebaut, an welchem der Splitpunkt von den Capsules gar nicht erreicht werden kann, das heisst, die SplitKnoten liegen nicht auf dem ordentlichen gerouteten Weg zwischen Sender und Empfänger. Die Capsules werden nun munter vorwärtsgerouted auf der Suche nach dem Split-Knoten. Dabei können diese bei ihrer "Suche" aber auch schon als unverschlüsselte Information bei den Knoten landen, welchen man nicht traut. Dies wäre ein gravierendes Problem, da der eigentliche Zweck des Systems, also die Verschlüsselung von Daten an bestimmten Stellen und das Senden dieser nicht mehr erfüllt wird.

- Ein drittes Szenario könnte wie folgt aussehen: Man konfiguriert sich einen Weg, auf dem sich die Teilwege auf dem gesplittete Capsules ver-

kehren kreuzen. Nach dem Splitten werden die Daten weitergerouted, doch Anhand der Routingtable beschliesst das System die Daten von diesem SplitKnoten an den anderen Splitknoten weiterzuleiten. Das System funktioniert, da die Daten wieder an einem bestimmten Punkt zusammengefügt werden und der Empfänger die kompletten Daten erhält. Jedoch werden das verschlüsselte Paket als auch der dazu passende Schlüssel auf dem gleichen Weg transportiert. Die Knoten welchen man nicht traut könnten, nun da sie die kompletten Informationen haben, an die unverschlüsselten Daten gelangen.

Allgemein entstehen auch unvorhersehbare Probleme, da Knoten ausfallen können und ein richtig funktionierendes System plötzlich in ein System der gezeigten Szenarien wechseln könnte. Da jedoch ANTS nicht dynamisches Routing unterstützt, sondern mit fest vorgegebenen Routingtables arbeitet, ist dieses Problem für die bestehende Implementation nicht relevant. In einer richtigen Umgebung, einer sogenannten "real-live" Implementation mit ggf. anderen Frameworks, muss man sich mit dieser Problematik unbedingt auseinandersetzen.

3.6.2 Mögliche Lösungen der gezeigten Probleme

- Einen Teil der Probleme könnte man umgehen, in dem man durch die Pathfinder-Capsules eine Bestätigung erhält, ob diese einen Split- resp. Merge- Knoten erfolgreich aufsetzen konnten. Somit wissen wir bereits, dass die entsprechenden Knoten wirklich erreichbar sind. Das Problem von Szenario 2 wäre somit teilweise gelöst.

- Zur Vermeidung des Problems in Szenario 3 muss man sich beim Aufsetzen des Pfades sehr genau überlegen, wie die Netzwerkstruktur aussieht, und wie entsprechende Pakete an den einzelnen Knoten weiter gerouted werden. Oder man kreiert sich eine neue Klasse von Paketen, welche man ans Ziel sendet und an den entsprechenden Knoten eine Markierung hinterlässt, wobei verschlüsseltes Paket und ein Schlüssel-Paket verschiedene Markierungen verwenden. Trifft nun ein solches Kontrollpaket auf eine Markierung ihres Gegenstücks, so weiss es nun, dass sich die aufgesetzten Teilwege der gesplitteten Pakete kreuzen und kann eine Nachricht an den Sender schicken und ihn über die Ungültigkeit des gewählten Pfades informieren.

- Analog wäre auch denkbar Testspakete zu erstellen, welche den aufgesetzten Pfad auf Typen vom Szenarien 1 und 2 überprüfen.

- Eine mögliche Implementation, welche einen Teil der beschriebenen Probleme gar nicht erst aufkommen lässt, verlangt das Aufsetzen des kompletten Pfades. Es werden also nicht nur die Split- und Merge-Punkte gesetzt, sondern auch die normalen Forwardingknoten explizit

konfiguriert. Dies bedeutet jedoch einen sehr hohen Konfigurationsaufwand im Falle von grossen und komplexen Pfaden. Zum Aufsetzen von kleinen Pfaden mit wenig Knotenpunkten ist dies jedoch ein sehr sicherer Ansatz. Leider ist aber auch diese Variante nicht vor Problemen gefeit, da ein Ausfall eines Knotens den ganzen Pfad lahm legen würde. Immerhin ist dies jedoch aus der Sicht der Sicherheit unkritisch, da einfach Pakete nicht mehr weitergerouted werden, jedoch nicht möglicherweise über unerwünschte Pfade, wie zum Beispiel in Szenario 3 erwähnt.

3.7 Weitere Möglichkeiten zur Verbesserung der Sicherheit

3.7.1 Verbesserung der zufälligen Schlüsselwerte in den Node-Caches

Zur Verbesserung der Sicherheit in den Node-Caches bezüglich den Schlüssel-Werten könnte man unter anderem folgende Möglichkeiten in Betracht ziehen:

- Die Schlüsselwerte könnten in einem separaten Thread alle X Sekunden neu generiert werden, resp. es könnte eine separate Schlüssel-Generatorklasse erzeugt werden, welche auch eine verbesserte Implementierung eines Zufallszahlenalgorithmus verwendet, als denjenigen in Java eingebauten Mechanismus zu verwenden. Problematisch dabei ist nur die genaue Synchronisierung zwischen den einzelnen Capsules (wird durch eine Pathfinder Capsule ein neuer Weg initialisiert, d.h. es werden neue Schlüssel-Werte generiert, so müssen die darauf folgenden Normal und Splitted Capsules auch informiert werden. Zu diesem Problem gibt es zum Glück eine einfache Lösung. Die offizielle Lebensdauer eines Eintrags ist ein wenig kürzer als diese dann effektiv in den Nodecaches verweilen werden. Somit werden auch noch Capsules, welche kurz vor dem Ablaufen der offiziellen Lebensdauer der letzten gültigen Schlüsselwerte, richtig durch das Netz gerouted, da nicht vorherzusehen ist, wie Lange eine Capsule vom Sende bis zum Empfängerknoten braucht (es gilt hier einen vernünftigen Mittelwert zu wählen).

- Weiterhin könnte anstelle eines Offsets, für alle Schlüsselwerte ein erneuter Zufallswert erzeugt werden.

- Anstelle eines Integers, könnte man einen Schlüssel als String beliebiger Länge, resp. ein Java-Objekt beliebigen Typs vorsehen (man könnte zum Beispiel direkt eine Art Verschlüsselungs-Objekt als Schlüsselwert abspeichern).

3.7.2 Verbesserung der zufälligen Werte für die Identifikation eines Nodes

Analog zu den Schlüssel-Werten, kann auch die Erzeugung für die Werte, welche einen Node als Split- oder Merge-Node identifizieren mit einem besseren Zufallszahlengenerator erzeugt werden, resp. in einem besseren Format abgespeichert werden. Auch ist eine zeitweise Erneuerung dieser Werte (im gleichen Rahmen wie die zugehörigen Schlüssel-Werte von Vorteil). Siehe dazu auch Kapitel 3.7.1.

3.7.3 Verschlüsselung der Werte für die Zieladressen

Die 2 Zieladressen, welche in einem Split-Node-Cache gespeichert werden, sind dort im Klartext abgespeichert. Man könnte diese jedoch auch verschlüsselt abspeichern und den Verschlüsselungsalgorithmus periodisch oder sogar in zufälligen Zeitabständen austauschen.

3.7.4 Fazit der möglichen Erweiterungen zur Sicherheit

Es gibt sicherlich noch weitere Möglichkeiten zur Verbesserung der Sicherheit. Aber auch schon bei den oben erwähnten Punkten muss nicht vergessen werden, dass ein gutes Gleichgewicht zwischen Performance und Sicherheit gewählt werden sollte. Was nützt einem die beste Sicherheitsstufe mit einer quasi komplett neuen Initialisierung von Zufallszahlengeneratoren und Verschlüsselungsalgorithmen nach X gesendeten Capsules oder einer Zeitdauer X, wenn es danach völlig uninteressant ist das SplitPad-Set zu benutzen, da die Performance nur gerade ausreicht kurze Texte zu senden, anstelle von ganzen Plänen als Grafiken oder gar kompletten Programmen?
Hier ist es also wichtig sich zu Überlegen, was die Anforderungen an das System sind. Wird es nur zum Versenden kurzer Mitteilungen von grosser Wichtigkeit gebraucht ist ein hoher Verschlüsselungsfaktor und eine möglichst kurze Lebensdauer der einzelnen Elemente zur Sicherheit sicherlich von Vorteil.
Sollten jedoch auch grössere Dateien oder nicht allzu kritische Informationen häufig durch das Netz gesendet werden, so ist eine einfachere, performanceoptimierte Verschlüsselungs- und Erneuerungs-Strategie sicher zu bevorzugen.

3.8 Performancemessungen an einem konkreten Beispielaufbau

Es wurde ein Beispielaufbau der zweiten Variante (also das Senden einer Datei) wie folgt getestet:

- Beispielnetzwerk mit 6 Nodes

- die Verschiedenen Nodes wurden in verschieden Subnetzen aufgestellt, welche zum Teil zwischen verschieden Routern liegen
- für die Split- und Merge-Nodes wurden die perfomanteren Maschinen ausgesucht

Es folgt die Konfigurations-Datei für ANTS:

```
# prepare a network with two paths form host to dest.
# the network topology is distributed over different hosts
# node ip as long start from 1000000000.
#
#     4
# 1-2< >5-6
#     3
#
# node 1 is sender
# node 2 is a split node
# node 3 and 4 are normal forwarding nodes
# node 5 is a merging node
# node 6 is the target node, the receiver

# node 1, sender, on localhost
node 59.154.202.1 -routes spfd.routes -log 255
channel 59.154.202.1 localhost:8001 -log 255
application 59.154.202.1 apps.SPAppFileSender -target 59.154.202.6
manager 59.154.202.1 -gui true -log 255

# node 2, split node, on alf
node 59.154.202.2 -routes spfd.routes -log 255
application 59.154.202.2 apps.SPAppFileNode
channel 59.154.202.2 alf:8002 -log 255
manager 59.154.202.2 -gui true -log 255

# node 3, normal forward node, on jerry
node 59.154.202.3 -routes spfd.routes -log 255
channel 59.154.202.3 jerry:8003 -log 255
application 59.154.202.3 apps.SPAppFileNode
manager 59.154.202.3 -gui true -log 255

# node 4, normal forward node, on paulchen
node 59.154.202.4 -routes spfd.routes -log 255
channel 59.154.202.4 paulchen:8004 -log 255
application 59.154.202.4 apps.SPAppFileNode
manager 59.154.202.4 -gui true -log 255

# node 5, merge node, on balu
node 59.154.202.5 -routes spfd.routes -log 255
channel 59.154.202.5 balu:8005 -log 255
```

```
application 59.154.202.5 apps.SPAppFileNode
manager 59.154.202.5 -gui true -log 255

#node 6, receiver, on asterix
node 59.154.202.6 -routes spfd.routes -log 255
channel 59.154.202.6 asterix:8006 -log 255
application 59.154.202.6 apps.SPAppFileReceiver -target 59.154.202.1
manager 59.154.202.6 -gui true -log 255

# the inter connections between the different virtual nodes
connect 59.154.202.1 59.154.202.2
connect 59.154.202.2 59.154.202.3
connect 59.154.202.2 59.154.202.4
connect 59.154.202.3 59.154.202.5
connect 59.154.202.4 59.154.202.5
connect 59.154.202.5 59.154.202.6
```

Listing: spfd.config

Danach wurde eine Datei mehrmals gesendet und die Log-Dateien mithilfe von GNUPlot graphisch aufbereitet:

3.8.1 Analyse der Daten des Sendeknoten

In der Abbildung 12 auf Seite 35 kann man beobachten, dass die meisten Capsules den Sendeknoten quasi ohne Verzögerung verlassen. Die entsprechenden Ausreisser sind auf interne, nicht kontrollierbare Abläufe in der Java Virtual Machine zurückzuführen.

3.8.2 Analyse der Daten des Splitknoten

Der Split-Knoten verhält sich nicht unbedingt wie erwartet. Die Werte des Delay liegen zwischen 1 und 700 ms. Siehe dazu Abbildung 13 auf Seite 36. Diese grosse Streuung ist auch wieder auf die internen Abläufe der Java Virtual Machine zurückzuführen. Bei einer kleinen Anzahl zu sendender Capsules, werden die einzelnen Capsules mit Verzögerungen zwischen 1- 35 ms verarbeitet. Sobald jedoch die Anzahl zu sendender Capsules zunimmt, ist die Java VM scheinbar überfordert und kann nur noch mit beschränkter Geschwindigkeit die Capsules abfertigen. Die genaue interne Ursache in der Java VM konnte aber leider nicht ausfindig gemacht werden.
Interessant zu Beobachten ist, dass sich eine leichte Grundlinie des Sendedelays abzeichnet, welche den wirklichen Delay im Netzwerk wiederspiegelt. Am Ende solcher Abschnitte in der Grundlinie steigt der Delay langsam

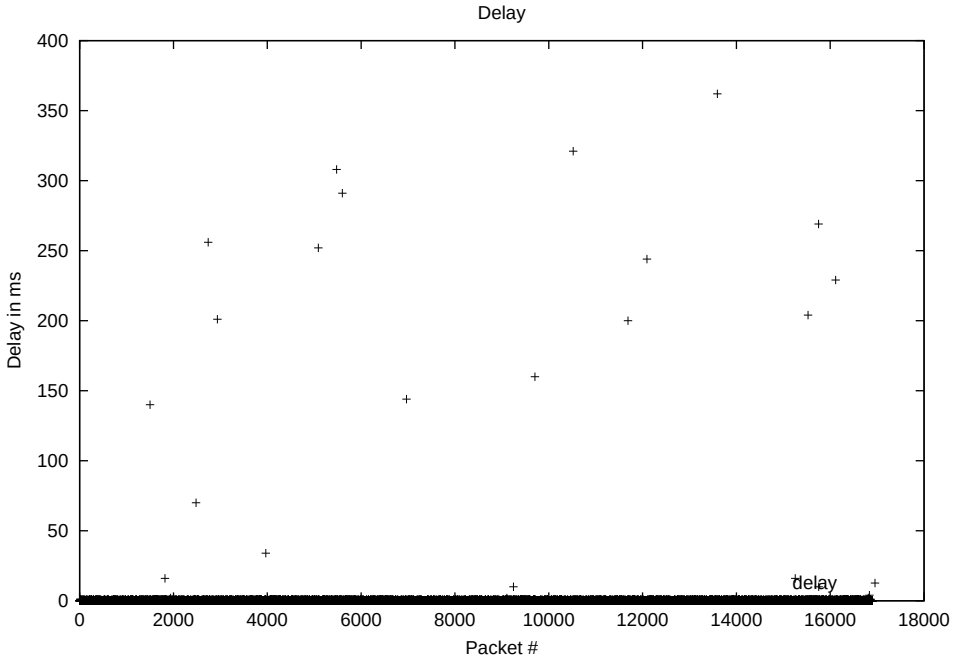

Abbildung 12: Grafische Darstellung des Delays im Sende-Knoten

an (Interrupt der Java VM, Garbage Collecting, etc.), bis dieser schliesslich wieder abrupt zur Grundlinie zurückkehrt.

3.8.3 Analyse der Daten des 1. normalen Forwardingknoten

Abbildung 14 auf Seite 36. Verhält sich wie der Split-Knoten, in Kapitel 3.8.2 beschrieben. Man kann jedoch beobachten, dass die Grundlinie des Sendedelays ein wenig weiter nach oben veschoben wurde, als im vorherigen Knoten.

3.8.4 Analyse der Daten des 2. normalen Forwardingknoten

Abbildung 15 auf Seite 37. Verhält sich wie der Split-Knoten, in Kapitel 3.8.2 beschrieben. Man kann jedoch wieder beobachten, wie die Grundlinie des Sendedelays ein wenig weiter nach oben veschoben wurde, als im vorherigen Knoten.

3.8.5 Analyse der Daten des Mergeknoten

Abbildung 16 auf Seite 38. Verhält sich wie der Split-Knoten, in Kapitel 3.8.2 beschrieben.
Auch hier lässt sich wieder beobachten, wie sich die Grundlinie des Sendedelays ein wenig weiter nach oben veschiebt, als im vorherigen Knoten.

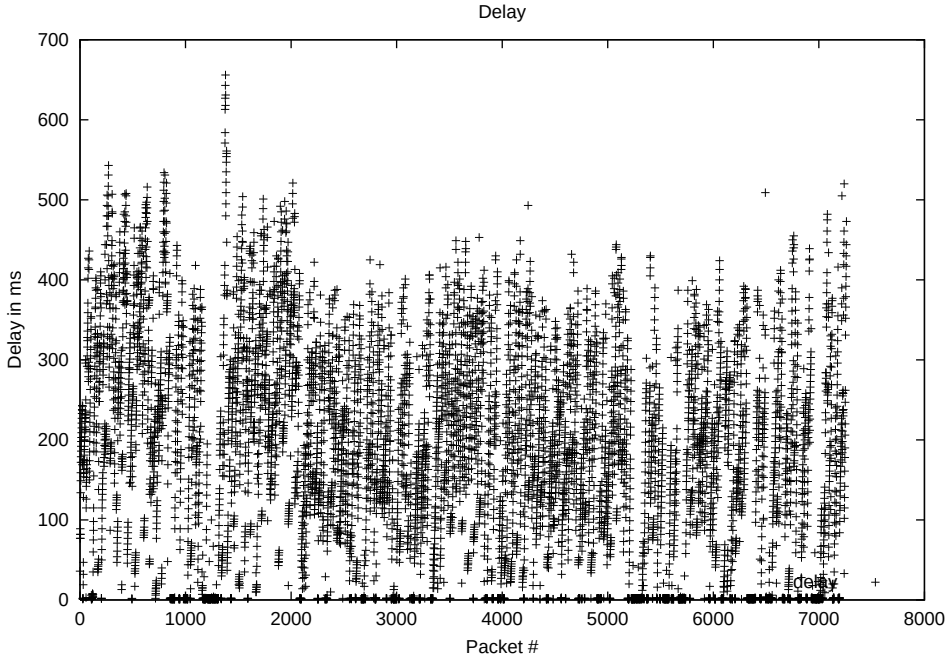

Abbildung 13: Grafische Darstellung des Delays im Split-Knoten

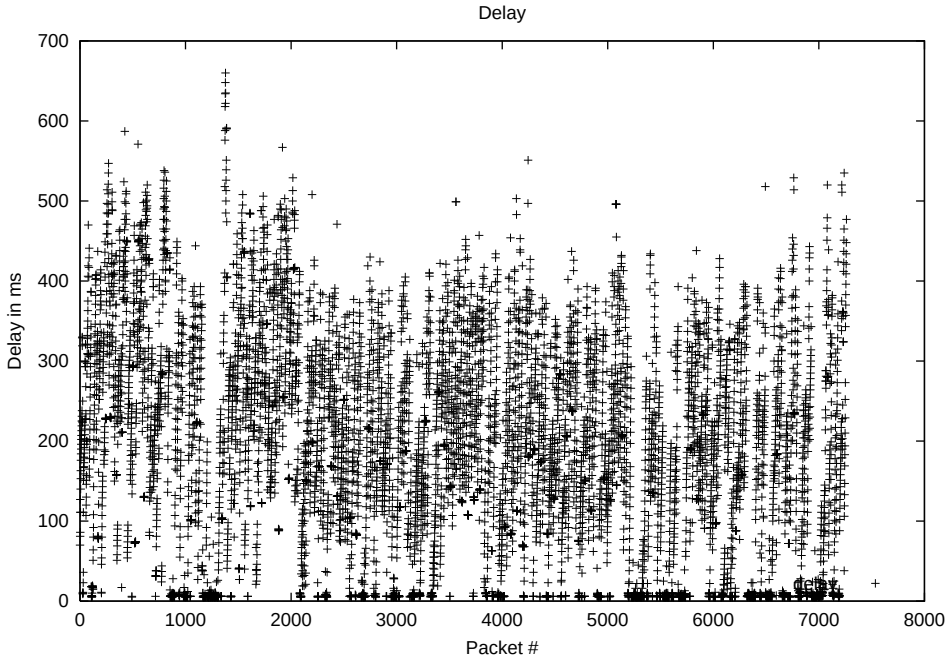

Abbildung 14: Grafische Darstellung des Delays im 1. Forward-Knoten

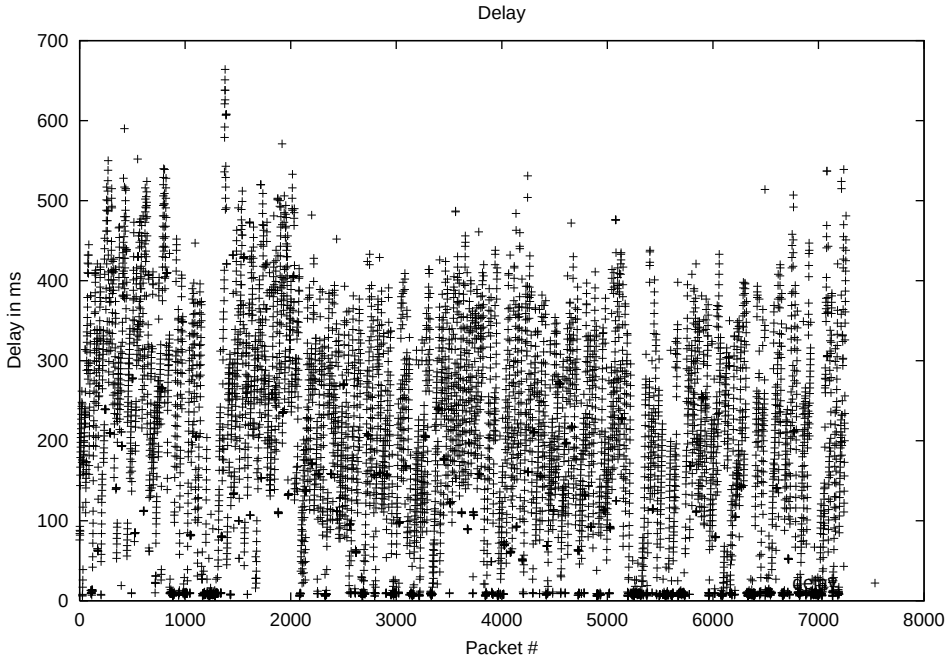

Abbildung 15: Grafische Darstellung des Delays im 2. Forward-Knoten

Zusätzlich jedoch sieht man hier, dass doppelt so viele Pakete wie in den sonstigen Knoten (ausser Sende-Knoten) verarbeitet wurden. Der Merge-Knoten erhält die gesplitteten Pakete von 2 verschiedenen Knoten zugesandt.

3.8.6 Analyse der Daten des Empfangsknoten

Abbildung 17 auf Seite 38. Verhält sich wie der Split-Knoten, in Kapitel 3.8.2 beschrieben. Auch hier gilt die Beobachtung, dass sich die Grundlinie des Sendedelays ein wenig weiter nach oben veschoben hat, als im vorherigen Knoten.

3.8.7 Analyse des Randomseed

In der Abbildung 18 auf Seite 39 sieht man, dass das Randomseed gleichverteilt ist. Weitere Informationen zur Beurteilung der Qualität des Randomseeds findet man in [GBB].

3.9 Fazit

Die Arbeit mit ANTS hat grösstenteils Spass gemacht, da man durch die Verwendung von Java und den bereits vorhandenen Beispielklassen und den Superklassen sehr schnell seine Ideen umsetzen kann.

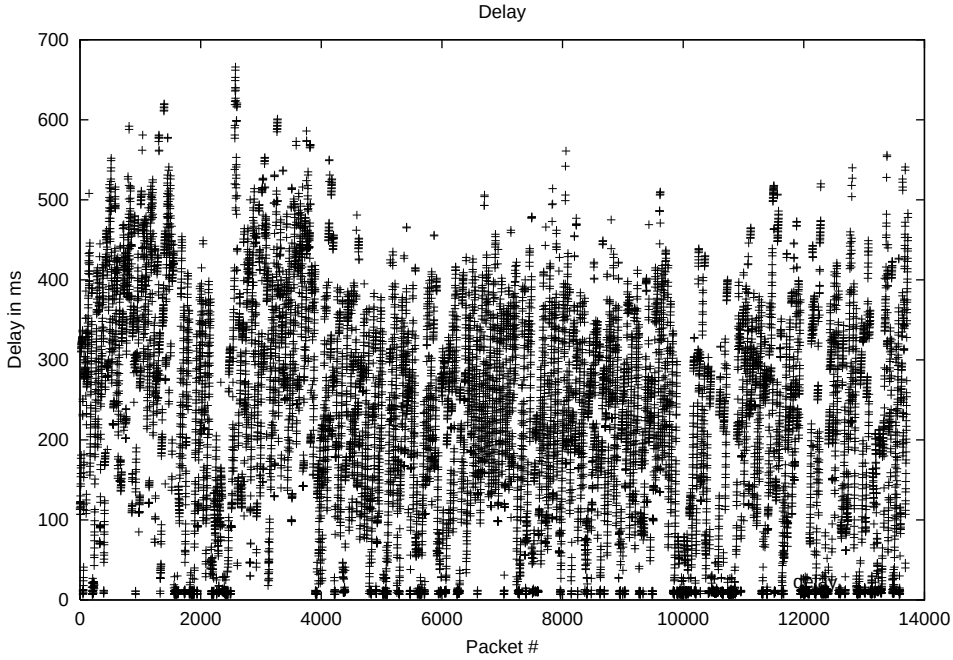

Abbildung 16: Grafische Darstellung des Delays im Merge-Knoten

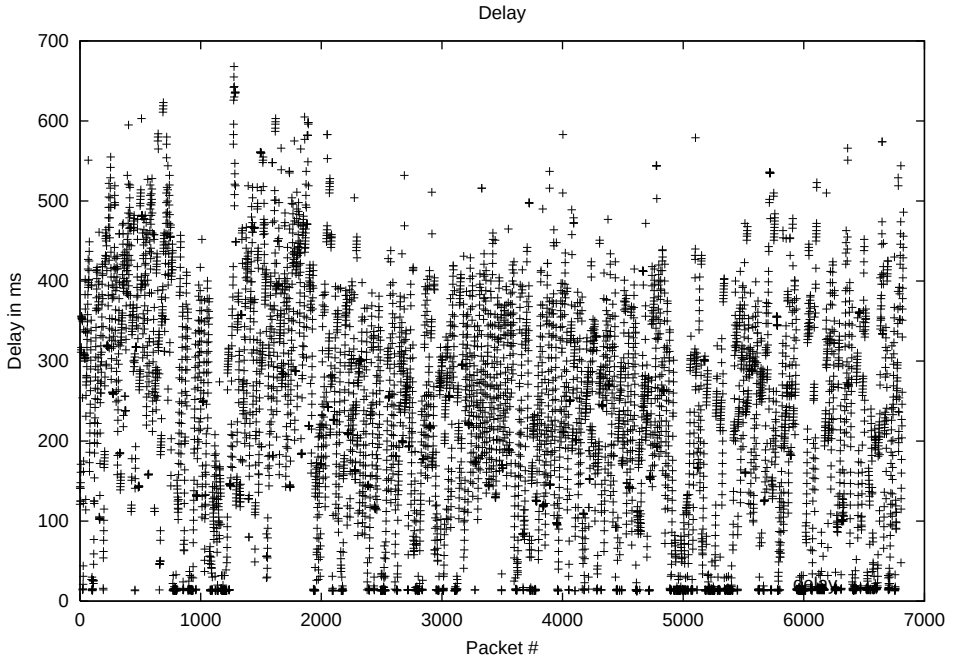

Abbildung 17: Grafische Darstellung des Delays im Empfangs-Knoten

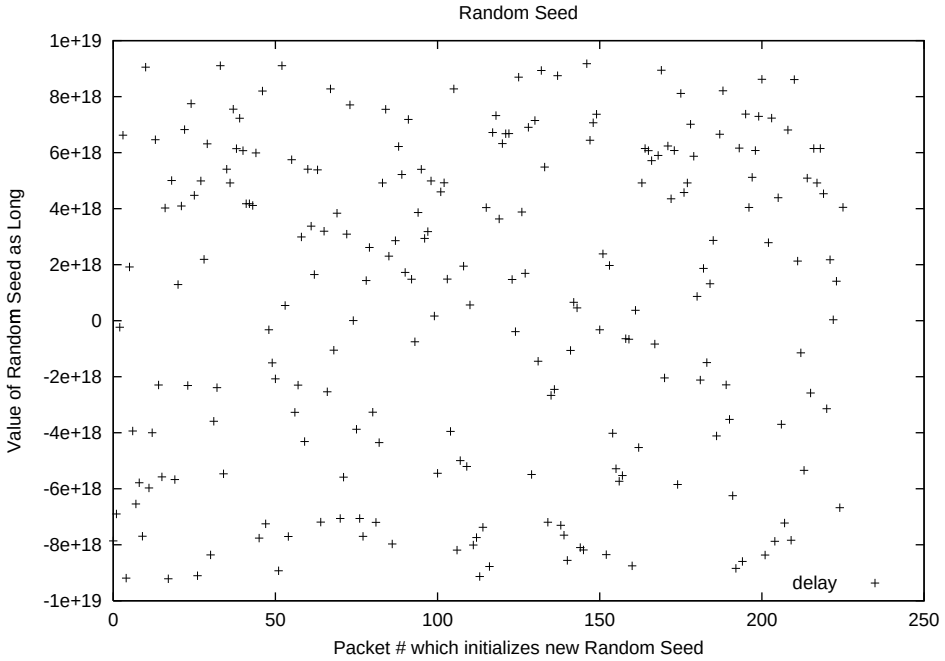

Abbildung 18: Grafische Darstellung des Verteilung des Randomseed

Leider ist jedoch mit der Verwendung von Java auch das Problem der schlechten Performance mitgeschleppt worden. Wie vorher gezeigt wurde, ist bei einer grossen Anzahl zu sendender Capsules das System schnell überfordert. Auch kann man nie genau wissen, was die Java Virtual Machine im Hintergrund alles so treibt und sie einem die Prozessorzeit in wichtigen Momenten plötzlich wegschnappen kann. Zeitkritische Anwendungen lassen sich damit sicher nicht zufriedenstellend realisieren.
Man kann jedoch mit Garantie sagen, dass sich ANTS zum Umsetzten vieler Ideen aus dem Bereich des Active Networking eignet, sei es nur zur Implementierung eines Prototypen oder zu Testzwecken der Realisierbarkeit einer Idee (dann auch für performancelastige Anwendungen).
Ein grosses Manko von ANTS ist jedoch die fehlende Dokumentation. Zwar gibt es ein paar interessante Beispiele die mitgeliefert werden, jedoch ist der Einarbeitungsaufwand, da sowohl die Beispiele als auch der Sourcecode des Frameworks gar nicht oder nur spärlich dokumentiert sind, sehr hoch. Einmal das Konzept begriffen und sich in ANTS eingearbeitet, lassen sich jedoch wie gesagt, sehr schnell damit Ideen in einer Implementation konkretisieren. Die Umsetzung der SplitPad-Idee mit ANTS zeigt, dass diese realisierbar ist und gut funktioniert. Sie wurde mit kurzen Mitteilungen als auch beim Versenden von grossen Dateien und Datenmengen getestet. Für eine kommerzielle Implementierung müsste jedoch sicherlich auf ein anderes Active Networking Framework zurückgegriffen werden.

Die weitere Entwicklung von ANTS hat seit dem Februar 2000 neuen Auftrieb erhalten. Im April 2000 hat David Wetherall[5] eine Version 1.3 des ANTS-Toolkit vorgestellt. Weitere Informationen zu der neuen Version findet man auf seiner neuen ANTS-Webseite[6].
Desweiteren existieren diverse Erweiterungen von ANTS. Projekte wie z.B. Magician[7] oder PANAMA[8] mit dem Unterprojekt AER/NCA[9] bauen mehr oder weniger auf ANTS auf resp. erweitern die Basisklassen von ANTS.

[5] http://www.cs.washington.edu/homes/djw/
[6] http://www.cs.washington.edu/research/networking/ants/
[7] http://www.ittc.ukans.edu/ kulkarn/projects/Magician.html
[8] http://www.tascnets.com/panama/
[9] http://www.tascnets.com/panama/AER/index.html

Literatur

[CSAA98] Maria Calderon, Marifeli Sedano, Arturo Azcorra, and Christian Alonso. Active Network Support for Multicast Applications. *IEEE Network*, 12(2):46–52, May/June 1998.

[GBB] Manuel Günter, Marc Brogle, and Thorsten Braun. Secure Communication: a New Application for Active Networks. Institute of Computer Science and Applied Mathematics, University of Berne, Neubrückstrasse 10, CH-3012 Bern, Switzerland.

[Wet97] David Wetherall. Developing Network Protocols with the ANTS Toolkit. Technical report, Massachusetts Institute of Technology (MIT), 08 1997. Auch gespeichert im Verzeichnis docs des ANTS-Toolkit als programming.ps.gz.

[WGT98] David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. Technical report, Massachusetts Institute of Technology (MIT), www.sds.lcs.mit.edu/publications/openarch98.html, 04 1998. Auch gespeichert im Verzeichnis docs des ANTS-Toolkit als architecture.ps.gz.

[WGT99] David Wetherall, John Guttag, and David Tennenhouse. ANTS: Network Services without the Red Tape. *Computer*, 32(4):42–48, April 1999.

[WLG98] David Wetherall, Ulana Legedza, and John Guttag. Introducing New Internet Services: Why and How. *IEEE Network*, 12(3):12–19, May/June 1998. www.sds.lcs.mit.edu/publications/network98.html.