Simulation einer Netzwerkumgebung für verteilte Paarprogrammierung

Bachelorarbeit

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

an der

Freien Universität Berlin
Fachbereich: Informatik
Studiengang: Bachelor Informatik

Betreuer: Barry Linnert
2. Prüfer: Prof. Dr. Margarita Esponda-Argüero

Eingereicht von
Timo Giesler
Berlin, 24.10.2016

Inhaltsangabe

1 Einleitung

Diese Arbeit handelt von der Entwicklung eines Programms zum Testen des Saros Plugins für verteilte Paarprogrammierung unter dem Einfluss verschiedener Effekte, die bei der Datenübertragung über das Internet auftauchen können.

1.1 Motivation

In der heutigen Softwareentwicklung wird die Zusammenarbeit mit Entwicklern, die sich nicht am selbem Ort befinden, immer wichtiger. Von Entwicklern aus verschiedenen Standorten der selben Firma bis hin zu Open-Source Projekten, die über die ganze Welt verstreute Mitglieder haben, das Arbeiten über größere Entfernungen hat in der Zeit des Internets eine große Bedeutung. Um die mit der Entfernung verbundenen Organisationsprobleme für die tägliche Arbeit zu bewältigen gibt es mehrere Programme, die dem Entwickler dabei helfen. Eines davon ist das Saros-Plugin für die Eclipse Entwicklungsumgebung, welche den Prozess der Paarprogrammierung über ein Netzwerk ermöglicht. Bei diesem Prozess geht es um die Zusammenarbeit mehrerer Entwickler, üblicherweise einem Programmierer und mehrere Beobachter, die an der selben Codestelle gleichzeitig zu Zwecken der Schulung oder der frühen Fehlererkennung arbeiten.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es, ein Programm zu entwickeln, das auf einem Gateway in der Testumgebung für das Plugin läuft und verschiedene Netzwerkeffekte simuliert um das Plugin auf Probleme beim Arbeiten mit einer nicht idealen Verbindung zwischen Client und Server zu testen. Dafür soll das Programm die über das Netzwerk gesendeten Daten entgegen nehmen, manipulieren und dann weiter leiten unter möglichst minimaler Änderung an den Clients oder am Server. Das Programm soll die gewünschten Manipulationen aus einer von der Testumgebung bereit gestellten Konfigurationsdatei einlesen, sodass diese vom Tester je nach Bedarf angepasst werden können, um verschiedene Netzwerkeffekte zu simulieren. Da der Test durch verschiedene Phasen läuft und nicht alle Phasen wie z.B. die Initialisierung davon beeinträchtigt werden sollen, muss das Programm eine Möglichkeit haben, auch verschiedene Phasen zu definieren, zwischen denen dann während des Test gewechselt werden kann, um die Manipulationen für unbestimmt lange Zeiträume zu verändern oder auszusetzen. Befindet sich keine Konfigurationsdatei auf dem Gateway, wird der Datenverkehr ungehindert weitergeleitet, um die bereits bestehenden Tests nicht zu beeinflussen.

2 Saros und Paarprogrammierung

Paarprogrammierung (engl. Pair Programming) bezeichnet eine Technik bei der mindestens zwei Programmierer zusammen an ein und demselben Quellcode gleichzeitig arbeiten. Dazu gibt es meist eine Person, die tatsächlich programmiert, während andere ihm dabei zuschauen und seinen geschriebenen Code verfolgen. Dies dient entweder zur Schulung neuer Programmierer, die durch das Beobachten von weiter fortgeschrittene Programmierer ihr Verständnis des Codes und allgemeinerer Programmiertechniken erweitern sollen, oder zur Fehlervorbeugung, indem die zweite Person die geschriebenen Code direkt auf mögliche Probleme kontrollieren können und so Fehler finden, die normalerweise erst in einem späteren Schritt erkannt werden würden. Die Vorteile von Paarprogrammierung liegt in der zeitnahen Untersuchung des Codes und der direkten Kommunikation zwischen Programmierer und Beobachter, welche auch viele Verständnisprobleme vorbeugt und frühzeitig eventuelle Probleme erkennen lässt. Dies führt zu einer geringeren Anzahl an Fehlern im Code, einer engagierteren Arbeitsweise der Beteiligten durch den direkten Kontakt sowie einer Förderung der Teamfähigkeit der Beteiligten. Zu beachten ist aber auch, dass nun mehr Personen für die Arbeit an einem Abschnitt benötigt werden, die sowohl zeitlich als auch örtlich koordiniert werden müssen.

Mittels verteilter Paarprogrammierung (engl. Distributed Pair Programming) versuchen Softwareentwickler zu mindestens die Notwendigkeit für den Aufenthalt am selben Ort zu umgehen. Dabei könne die Beteiligten an verschiedenen über ein Netzwerk verbundenen Computern arbeiten, indem Sie spezielle Editoren benutzen, die den Code abgleichen und somit eine Beobachtung über beliebige Entfernungen ermöglichen. Das Programm, das in unserem Fall benutzt wird, ist das Saros Plugin, welches unter anderem für die Eclipse Entwicklungsumgebung existiert. Dieses synchronisiert die Editor Ansicht zwischen den einzelnen Client, indem Änderungen an einen normalen XMPP Server gesendet werden, der die Daten dann einfach an alle anderen Clients, die für diese Sitzung angemeldet sind, weiterleitet. Sollte es zu größeren Änderungen kommen, kann die ganze Datei auch direkt von Client zu Client geschickt werden.

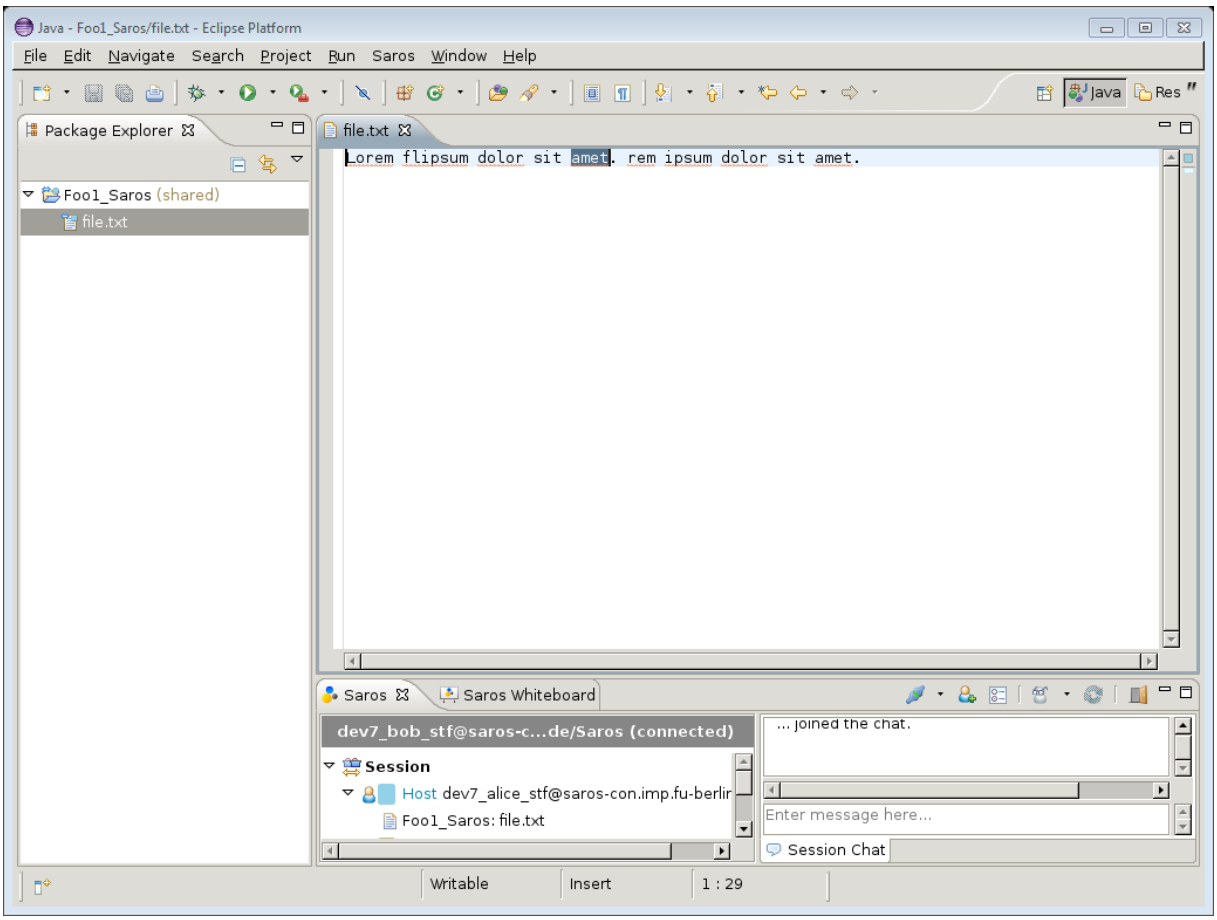

Abb. 1 Eclipse mit Saros Plugin und einem geteilten Projekt.
Verbindungsinformationen und Chat-Fenster im unteren Bereich.

Der Nachteil dieser Arbeitsweise ist sowohl der Wegfall der direkten Kommunikation zwischen dem Programmierer und den Beobachtern als auch der negative Einfluss von Netzwerkeffekten auf den Datenverkehr. Für die Kommunikation stellt das Plugin zwar eine Chat-Applikation zur Verfügung, es wird aber empfohlen zusätzlich eine Audioverbindung via VoIP oder ähnlichem herzustellen. Sollte es jedoch zu stärkeren Verbindungsproblemen kommen oder einer der Beteiligten nicht in der Lage sein, eine Verbindung mit dem Internet herzustellen, führt dies zum Ausfall des Programms.

3 Testaufbau

Zum Testen des Plugins bedienen sich die Entwickler mehrere virtueller Maschinen (VM), auf denen die Clients je mit einer Saros Umgebung versehen werden. Zusätzlich gibt es noch eine virtuelle Maschine die als Gateway agiert und durch die die Daten der Clients und des Servers gesendet werden. Die VMs werden dann mittels eines Job-Management-Systems mit der entsprechenden Client Version und Konfiguration bespielt und entfernt gestartet, wo sie dann gegen automatisierte Testfälle geprüft werden. Der Vorteil der virtuellen Maschinen ist der, dass sie in kurzer Zeit ohne zusätzliche Hardware bereitgestellt werden kann und Anpassungen auch aus der Entfernung über ein Netzwerk getätigt werden können.

Im Laufe dieser Arbeit wurde der Aufbau der virtuellen Maschinen kopiert und anschließend mittels Bearbeitung der Netzwerkkonfiguration ein eigenes Gateway erstellt, durch das alle anwendungsrelevante Daten laufen und ein auf dem Gateway befindliches Programm in der Lage ist, diesen Datenverkehr zu beeinflussen. Zum Steuern des Programms wird die geforderte Konfigurationsdatei ebenfalls dort platziert, mit der Idee, das diese dann ebenfalls über das Job-Management-System vor dem Test ausgetauscht wird.

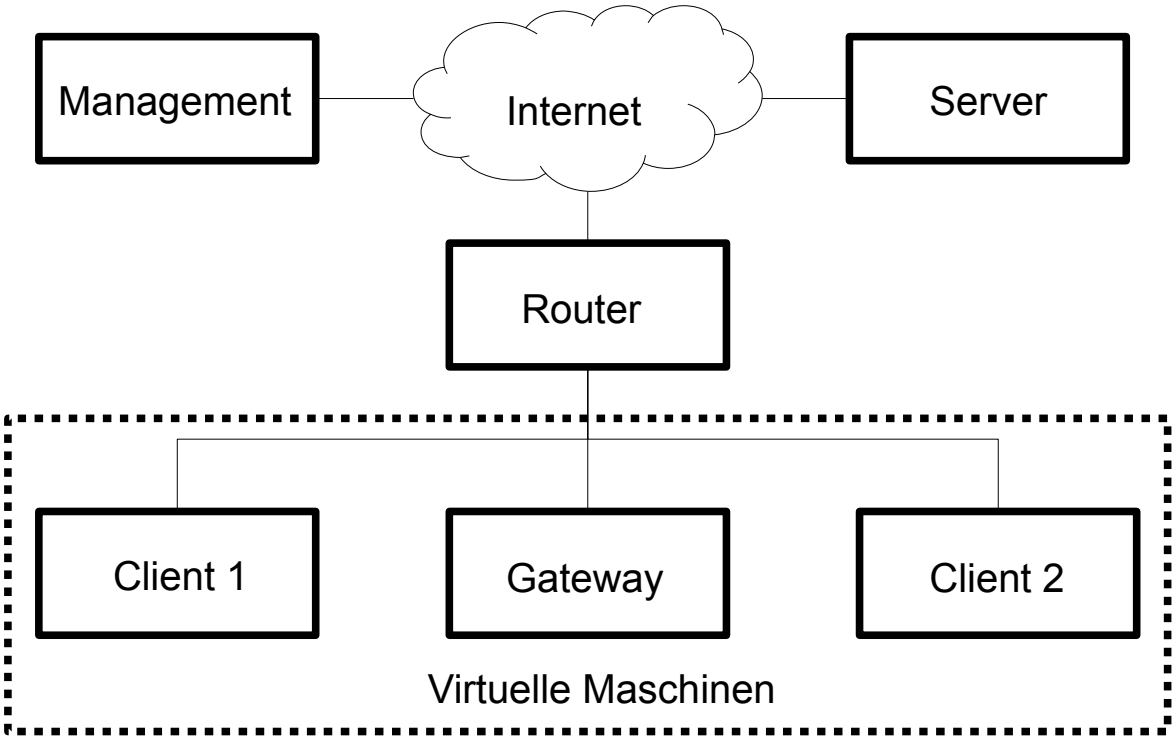

Abb. 2 Aufbau der Testumgebung

Um sicherzustellen, dass sämtliche Anwendungsdaten, welche als TCP Pakete verschickt werden, über das Gateway geleitet werden, hatte ich ursprünglich die IP-Adresse des Gateways bei den Clients als Standardgateway eingetragen. Nach wiederholten Abnormalitäten in der Verfolgung des Weges der Netzwerkpakete durch das System wurde jedoch klar, das die Routingprotokolle auf den Clients für Pakete zum Server bemerkt haben, das eine direktere Verbindung ohne Gateway direkt zum Route existiert. Daraufhin ignorierten nachfolgende Pakete das Gateway, was die eigentliche Idee des Gateways zu Nichte machte.

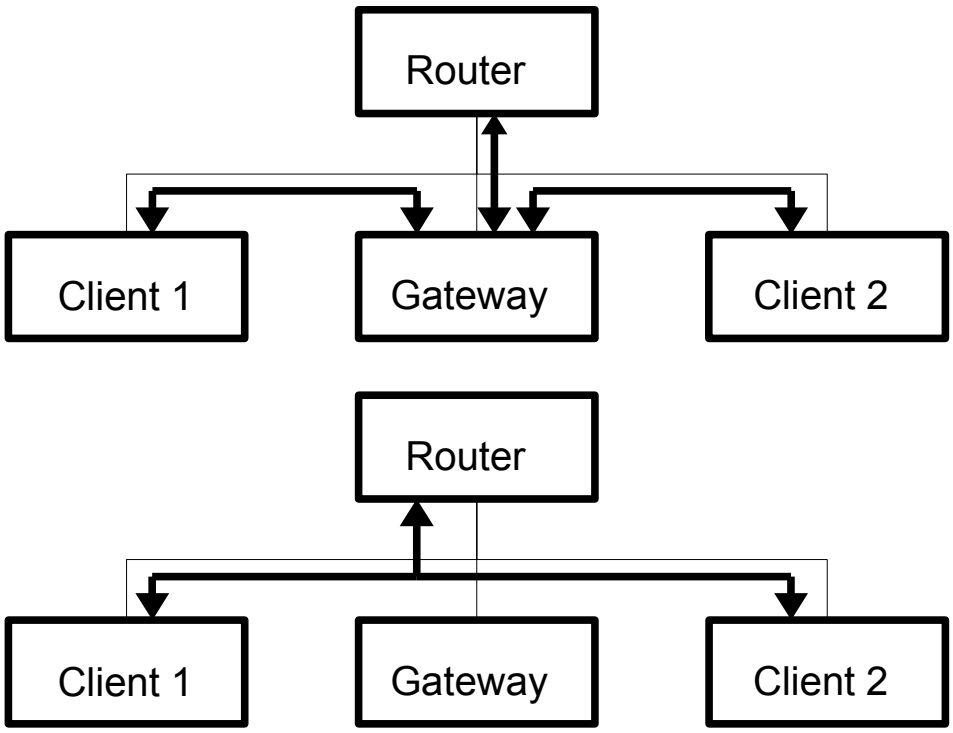

Abb. 3 oben: gedachte Routen, unten: tatsächliche Routen

Als Alternative habe ich zwei virtuelle LAN Netzwerke erzeugt, jeweils eins zwischen Client und Gateway, sodass die beiden Clients sich nun in zwei verschiedenen Netzwerken befinden und nur noch über das Gateway auf Ressourcen im Internet wie den Server zugreifen können. Für diesen Aufbau müssen auf den Clients lediglich die Netzwerkeinstellungen geändert werden, wovon die Arbeitsweise der Entwicklungsumgebung nicht beeinflusst wird. Um sicher zustellen, dass die Datenmanipulation auf dem Gateway nicht die Verbindung zwischen dem Job-Management-System und den Clients unterbindet, blieben die alten Verbindungen zwischen Client und Router mit einer niedrigeren Priorität bestehen. Dadurch kann das Gerät selber keine ausgehende Verbindung darüber herstellen, von außen kommende Verbindungen sind aber weiterhin möglich.

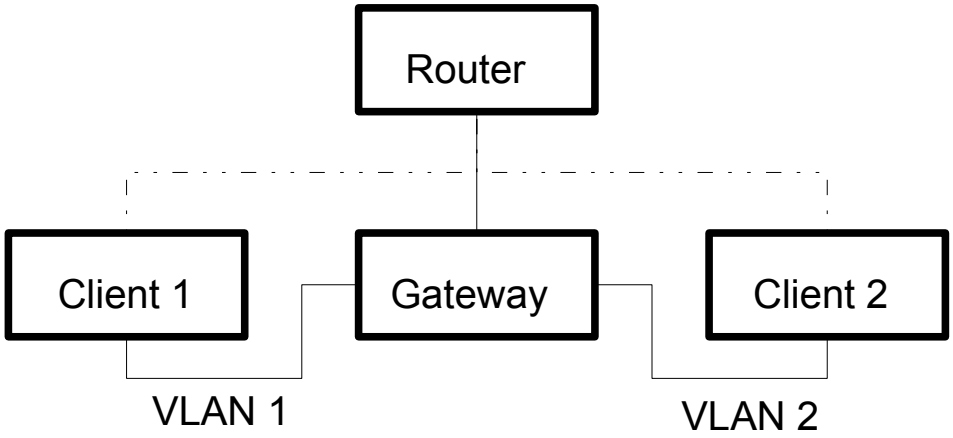

Abb. 4 Testaufbau mit zwei virtuellen Verbindungen

4 Verkehrskontrolle mit Netem

Für die Manipulation der Datenpakete benötigen wir einen möglichst in das Betriebssystem integrierten Zugriff auf diese Pakete, um den Verwaltungsaufwand so gering wie möglich zu halten und keine Änderungen an den Clients vorzunehmen.

4.1 Linux und Verkehrskontrolle

Auf dem zum Testen verwendeten Maschinen benutzen wir Debian, eine Distribution des Linux Betriebssystems, da dieses zum einen frei erhältlich ist und zum anderen auf Modulen basiert, die je eine oder mehrere Betriebssystemfunktionen erfüllen. Dies erlaubt es einem Entwickler Kernmodule nach Bedarf hinzuzufügen und zu entfernen, um so ein spezialisiertes System zu erstellen, was für viele Testumgebungen eine fast ideale Basis erzeugt.

Auf der Suche nach einem existierenden Modul, dass mir erlauben würde, die durch den Computer fließenden Datenpakete zu manipulieren, bin ich auf das Modul "Traffic Control" (z.Dt. Verkehrskontrolle) gestoßen. Dieses erlaubt es, die IP-Pakete, die in einer Warteschlange darauf warten von dem Computer über ein Netzwerk gesendet zu werden, mittels verschiedener Regeln zu beeinflussen. Das Modul ist über den Befehl tc steuerbar, indem als Parameter die betroffene Netzwerkschnittstelle und eine Regel aus den installierten Regelwerken mitgegeben wird.

```
tc qdisc [add|del|replace] <Netzwerkadapter> <Klasse> <Regelwerk> <Regel>
tc qdisc add eth0 root netem delay 20ms
```

Abb. 5 generischer TC-Befehl und Beispiel für die Verzögerung um 20ms

4.2 Regelwerke

Einer dieser Regelwerke ist TBF, kurz für Token Bucket Filter, die beschreibt, wie viele Bytes sich maximal in der Warteschlange befinden dürfen und mit welcher Geschwindigkeit diese die Warteschlange verlassen dürfen. Genauer dürfen Pakete nur dann gesendet werden, wenn dadurch die vorgegebene Gesamtgeschwindigkeit nicht übertroffen wird. Somit simuliert dies praktisch die Limitierung der Bandbreite.

```
tbf rate <Bandbreite> buffer <Buffergröße> limit <Warteschlangengröße>
tc qdisc add eth0 root tbf rate 500kbit buffer 1600 limit 3000
```

Abb. 6 generische TBF-Regel und Beispiel für die Limitierung auf 500 kbit

Ein weiteres Regelwerk ist Netem, kurz für Network Emulation, das mehrere Regeln bereit stellt, die bestimmte Effekte, deren Datenpakete in Netzwerken unterliegen, simulieren sollen. Dazu gehören zum einen Verzögerung und Umsortierung, die beide die sich in der Warteschlange befindlichen Pakete umzusortieren, um diese so erst mit einem bestimmten Zeitversatz oder mit einer geänderten Reihenfolge zu versenden. Für diese Befehle gibt man als Kommandozeilen-Parameter zusätzlich die Verzögerung und evtl. noch eine Variation und die Verteilung an, wie z.B. die Normalverteilung, nach der dann die endgültige Verzögerung berechnet wird. Zum Anderen gibt es noch die Effekte der Korruption, des Verlusts und der Vervielfachung der Pakete. Diese Befehle werden alle mit einer Wahrscheinlichkeit versehen und sorgen entweder für die Änderung des Inhalts des Pakets oder das Duplizieren oder das Wegwerfen des gesamten Pakets.

```
tc qdisc add eth0 root netem loss random 25%
tc qdisc add eth0 root netem duplicate 10%
tc qdisc add eth0 root netem delay 20ms 10ms distribution normal
```

Abb. 7 Netem-Regeln für den Verlust und Vervielfachung von Paketen sowie der Verzögerung um 20ms ±10ms mit Normalverteilung

Da das Saros Plugin seine Pakete über eine TCP-Verbindung verschickt und die TCP-Schicht solche Probleme automatisch behebt, haben diese im unseren Fall lediglich eine weitere Auswirkung auf die Verzögerung und die Übertragungsrate. Des Weiteren können alle Netem Regeln mit einer Korrelation versehen werden, welche beschreibt, wie stark die Effekte eines Paketes das Nächste beeinflussen. Dies führt zu einer wellenförmigen Zu- und Abnahme der Effekte bei hoher Korrelation.

Möchte man nun mehrere Regelwerke auf der selben Verbindung anwenden, um verschiedene Effekte gleichzeitig zu simulieren, bietet der Traffic Control Befehl für jede Netzwerkschnittstelle ein hierarchisches Klassensystem an. Jeder Klasse kann dabei eine Regel zugeordnet werden, die auch alle Unterklassen betrifft und so mehrere Regeln in der Unterklasse vereint. Für jeder dieser Klassen ist es dann auch möglich einen Filter anzulegen, der die von der Regel betroffenen Pakete anhand von Feldern im IP-Header, wie der Zieladresse und dem Port des Senders, bestimmt und alle anderen Pakete ignoriert.

```
tc qdisc add eth0 root handle 1:0 tbf rate 100kbit buffer 1600 limit 3000
tc qdisc add eth0 parent 1:0 handle 10 netem loss random 25%
```

Abb. 8 Einfache Klassenhierarchie für zwei Befehle

Da diese Regelwerke alle von uns gewünschten Manipulationen bereitstellen und mehrere kurze Tests auf unseren Maschinen ergeben haben, dass sie sich recht einfach über direkt messbare Werte wie Verzögerung, Verlust und Bandbreite steuern lassen und dass auch das benötigte Klassensystem keine größere Herausforderung darstellt, habe ich mich dafür entschieden, diese Modul die beschriebenen Regelwerke zur Entwicklung des Programms zu benutzen.

5 Saros Network Tester

Der Ansatz für das Programm, Saros Network Tester genannt, ist der, dass sich auf dem Gateway zwei Skripte befinden, die die Netzwerkemulation entsprechend starten bzw. beenden. Die Parameter für die einzelnen Regeln werden dabei aus einer Konfigurationsdatei gelesen und dann zu Kommandozeilen-Befehlen zusammen gesetzt, die letztendlich ausgeführt werden.

Für die Konfigurationsdatei habe ich mich für eine direkte Umsetzung der Netem- und TBF-Parameter in eine XML-Struktur entschieden, da diese textbasiert ist und so einfach per Hand erstellt oder bearbeitet werden kann und auch mit entsprechenden Bibliotheken aus dem Programm heraus eingelesen werden kann. Die Datei enthält dabei für den Server und die Clients je eine Konfiguration, bestehend aus den einzelnen Netem bzw. TBF Regeln als Tags und deren Parameter als Attribute.

```
<NetworkTest>
	<Connection name="Client1">
		<Delay duration="20ms"/>
	</Connection>
	<Connection name="Client2">
		<Corruption chance="5%" correlation="20%"/>
	</Connection>
	<Connection name="Server"/>
</NetworkTest>
```

Abb. 9 Erste Version der Konfigurationsdatei mit einfachen Regeln

Das eigentliche Programm an sich liest diese Datei dann ein, parst die einzelnen Elemente und wandelt sie in entsprechende Befehle um, die dann ausgeführt werden. Zur Sicherstellung der syntaktischen Korrektheit der Datei kann diese mit einem gegebenen XML-Schema geprüft werden.

Da die Testumgebung sehr langlebig ist und sich nur selten ändern, wurde entschieden eine weitere Datei manuell neben den Skripten auf das Gateway zu schreiben, die eine Zuordnung zwischen den Gerätebezeichner der Netzwerkkarten und selbst gewählte Namen der einzelnen Computern wie den Clients und dem Server enthält. Dies hat den Vorteil, dass zum einen die Konfigurationsdateien nur mit menschenlesbaren Name handeln müssen und zum anderen dass die Anwendungskonfiguration unabhängig von der direkten Netzwerkkonfiguration wird und so für mehrere Testumgebungen benutzt werden kann.

Um dem Programm nun die Möglichkeit zu geben, die simulierten Netzwerkeffekte nach bestimmter Zeit zu ändern, um es so auf Änderungen in einem Netzwerk zu testen, wurden die Parameter in der Konfigurationsdatei in einzelne Zeitslots gruppiert, die je mit einer zeitlichen Dauer markiert wurden.

```
<NetworkTest>
	<Connection name="Client1">
		<Timeslot duration="10m">
			<Bandwidth limit="100kbit"/>
		</Timeslot>
		<Timeslot duration="15s">
			<Corruption chance="5%" correlation="20%"/>
		</Timeslot>
		<Timeslot duration="infinite">
			<Loss chance="9%" correlation="15%"/>
		</Timeslot>
	</Connection>
</NetworkTest>
```

Abb. 10 Zweite Version der Konfigurationsdatei mit Zeitbegrenzungen

Das Programm wurde so abgeändert, dass es sich nicht mehr direkt nach dem Start der Manipulation der Pakete beendet, sondern dass es auf den Start der nächsten Zeitslots wartet und dann die alten Manipulationsregeln gegen die Neuen austauscht, bis es am Ende bei "permanenten" Zeitslots ankommt, die für den verbleibenden Rest des Test aktiv bleiben.

```
Einlesen der Konfigurationsdatei
Einlesen der Netzwerkkonfiguration
Entfernen alter Regeln
Solange noch begrenzte Zeitslots laufen:
      Wartezeit zurücksetzen
      Für jede Verbindung:
            Ermittlung des nächsten Zeitslot
            Austauschen der Regeln
            Wartezeit aktualisieren
      Auf nächsten Wechsel warten
```

Abb. 11 Grober Pseudocode des Skripts

Für die Realisierung der einzelnen Phasen des Durchlaufs gab es schließlich mehrere Zeitslots, die keine Zeitbegrenzung haben und so die einzelnen Zeitslots, die zu einer Phase gehören, von einander abtrennen. Die momentane Phase wird nun einfach in einer temporären Datei gespeichert und bei jedem Start ausgelesen bzw. beim Beenden der verschiedenen Tests zurückgesetzt, damit beim nächsten Durchlauf wieder von vorne begonnen wird.

```
<NetworkTest>
      <Connection name="Client1">
            <Timeslot duration="10m">
                  <Bandwidth limit="10mbit"/>
            </Timeslot>
            <Timeslot duration="infinite">
                  <Loss chance="10%"/>
                  <Bandwidth limit="100kbit"/>
            </Timeslot>
            <Timeslot duration="15s">
                  <Corruption chance="5% correlation="20%"/>
            </Timeslot>
            <Timeslot duration="infinite">
                  <Loss chance="9%" correlation="15%"/>
            </Timeslot>
      </Connection>
</NetworkTest>
```

Abb. 12 Dritte Version der Konfigurationsdatei mit mehreren Phasen

Bei der Wahl der Programmiersprache war hauptsächlich wichtig, dass das Programm auf den meisten Debian-Systemen möglichst ohne Installation zusätzlicher Programme auskommt. Zusätzlich sollte es eine entsprechende Code-Bibliothek zum Einlesen von XML-Dateien geben und das Programm auch ohne graphische Oberfläche auskommen, um die Anforderungen an die Maschine, auf dem das Gateway läuft, zu minimieren. Deshalb habe ich mich für Python entschieden, eine Skriptsprache dessen Interpreter auf vielen Debian-Systemen vorinstalliert ist.

6 Auswertung

Das Ziel des Programms, den Anwendungs-Datenverkehr mit dem Programm zu manipulieren, wurde großteils erfüllt. Das Programm kann mittels der Konfigurationsdatei gesteuert werden, die vom Tester individuell bereitgestellt werden kann. Bei älteren Tests, die keine Datei auf das Gateway legen oder Tests, die eine fehlerhafte Datei benutzen, beendet sich das Programm einfach und führt keine Manipulationen durch. Das Programm unterstützt dabei sowohl zeitliche Änderungen als auch den Wechsel zwischen mehreren Phasen.

Die einzige Änderung, die an der Netzwerkkonfiguration gemacht wurde, war die Erstellung und die Umleitung der Daten über die virtuellen Netzwerke. Da diese Änderung permanent ist und nur an den lokal veränderbaren Clients und dem Gateway durchgeführt werden muss, ist dies nicht problematisch für die Durchführung der Tests.

Bei der Arbeit mit dem Traffic-Controll-Modul ergab sich jedoch das Problem, dass die Manipulationen der Datenverbindungen nur auf vom Gateway ausgehenden Paketen funktioniert und die Pakete auf dem Weg zum Gateway ungehindert sind. Dies wurde für die von uns betrachteten Fälle als ausreichend erachtet und würde nur bei Problemen, die auf bestimmten Routen (z.B. von Client zu Client aber nicht von Client zu Server) auftreten, zu Schwierigkeiten führen.

7 Ausblick

Eine mögliche Verbesserung, die bei einer weitergehenden Entwicklung eingebracht werden kann, ist das Filtern der Manipulationen auf Pakete, die zwischen Client und Server sowie zwischen Client und Client gesendet werden. So kann verhindert werden, dass ein Fehler im Programm oder eine ungünstig erstellte Konfigurationsdatei den Zugriff von außen auf das Gateway unterbricht und so vom Management-System abschneidet.

Eine weitere Möglichkeit wäre ein komplizierterer Aufbau der VMs mit Clients und Gateway um nicht nur vom Gateway ausgehende Pakete, sondern alle Pakete zu beeinflussen und die Manipulationen so gründlicher durchzuführen.

8 Quellenangabe

FU-Logo: `https://upload.wikimedia.org/wikipedia/de/thumb/7/71/Fub-logo.svg/2000px-Fub-logo.svg.png`

Saros: `http://www.saros-project.org/`

TC: `http://man7.org/linux/man-pages/man8/tc.8.html`

TBF: `http://man7.org/linux/man-pages/man8/tc-tbf.8.html`

Netem: `http://man7.org/linux/man-pages/man8/tc-netem.8.html`

Saros Network Tester

Aufbau

In dem Ordner, in dem sich das Programm befindet, sollten folgende Dateien liegen:

main.py	Das Hauptprogramm
main.sh	Skript zum automatischen Aufrufen des Python-Interpreters
reset.py	Programm zum Beenden und Zurücksetzen
reset.sh	Skript zum automatischen Aufrufen des Python-Interpreters
devs.xml	Datei mit den Netzwerkinformationen
config.xml	Konfigurationsdatei für die Netzwerkeffekte
phase.txt	Enthält die momentane Phase, wird automatisch erstellt

Netzwerkinformation

Dies ist eine XML-Datei, deren Wurzelelement "Devices" heißt, der pro betroffenen Gerät ein "Device" Element hat, je mit den Attributen "name" für einen selbst gewählten Namen und "dev" für die Netzwerkschnittstelle.

```
<Devices>
	<Device name="Client1" dev="eth1"/>
	<Device name="Client2" dev="wlan2"/>
	<Device name="Server" dev="eth0"/>
</Devices>
```

Konfigurationsdatei

Dies ist ebenfalls eine XML-Datei, diesmal mit "NetworkTest" als Wurzelelement. Dieser hat für jede Verbindung ein Element namens "Connection" mit dem oben gewählten Namen als "name" Attribut. Hier drin befinden sich beliebig viele "Timeslot" Elemente, die je ein Attribut "duration" haben, die die Dauer des Zeitslots angeben. Ist der Wert "infinite", so ist dieser Zeitslot der Letzte seiner Phasen und bleibt bis zum Phasenwechsel oder zum Ende des Programms bestehen. Die Zeitslots enthalten nun die Regeln, jeweils mit den Parametern als Attribute.

Alle Regeln und deren Parameter:

Regel	Element	Attribute
Verlust	Loss	Wahrscheinlichkeit als "chance", evtl. Korrelation als "correlation"
Verdopplung	Duplication	siehe Verlust
Korruption	Corruption	siehe Verlust
Umsortierung	Reordering	siehe Verlust
Bandbreite	Bandwidth	Bandbreite als "limit"

Zusätzlich gibt es noch die Verzögerung als "Delay" mit dem Parameter "duration". Schwankungen können in dem Parameter "variation" angegeben werden. Gibt es Schwankungen, kann zusätzlich noch die Verteilung in Form der Korrelation als "correlation" oder in Form einer üblichen Verteilung als "distribution" mit den möglichen Werten "normal", "pareto" und "paretonormal".

Als Einheiten werden akzeptiert:

Minuten	m
Sekunden	s
Millisekunden	ms
Prozent	%
Bytes pro Sekunde	bps
Kilobytes pro Sekunde	kbps
Megabytes pro Sekunde	mbps
Kilobits pro Sekunde	kbit
Megabits pro Sekunde	mbit

```
<NetworkTest>
	<Connection name="Client1">
		<Timeslot duration="infinite">
			<Delay duration="20ms"/>
		</Timeslot>
		<Timeslot duration="infinite">
			<Delay duration="50ms" variation="10ms"/>
			<Bandwidth limit="100kbit"/>
		</Timeslot>
	</Connection>
	<Connection name="Client2">
		<Timeslot duration="15s">
			<Corruption chance="5%" correlation="20%"/>
		</Timeslot>
		<Timeslot duration="infinite">
			<Duplication chance="9%"/>
		</Timeslot>
		<Timeslot duration="infinite">
			<Loss chance="7%" correlation="30%"/>
		</Timeslot>
	</Connection>
	<Connection name="Server">
		<Timeslot duration="infinite"/>
		<Timeslot duration="infinite"/>
	</Connection>
</NetworkTest>
```

Aufruf

Das erste Aufrufen von main startet das Programm, alle Weiteren Aufrufe starten die jeweils nächste Phase. Der Aufruf von reset entfernt alle Netzwerkeffekte und setzt die Phase auf 0 zurück. Sollten eine der beiden Konfigurationsdateien fehlen beendet sich das Programm ohne etwas zu tun. Sollte einer der Verbindungen nicht genug Phasen definiert haben, bleibt das Programm bei der letzten funktionierenden Phase.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Datum: _____

Unterschrift: _____