UNIVERSITÄT KOBLENZ · LANDAU

Abteilung Koblenz
Fachbereich 4: Informatik

Implementierung des Spanning Tree Algorithmus IEEE 802.1D

Implementation of the spanning tree algorithm IEEE 802.1D

Vorgelegt von:

Andreas Sebastian Janke
andrjank@uni-koblenz.de
Matrikelnummer: 206210019

Betreuer:

Prof. Dr. Steigner
Dipl.-Inf. Frank Bohdanowicz

Koblenz, der 24.03.2010

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, den . . . . . .

Abstract

Im Rahmen dieser Bachelorarbeit wird ein Programm in Java entwickelt, mit dem man sich beliebige Netzwerke graphisch anzeigen lassen kann. Diese Netzwerke müssen im Vorfeld mit Hilfe eines Configuration-Files beschrieben werden und dürfen nur aus Layer 2 Switches und Hosts aufgebaut sein. Sollte man nun ein solches Configuration-File erstellt haben, wird man diese ins Programm laden können, wo dann das Netzwerk visualisiert wird. Darauf wird man dann den Spanning Tree Algorithmus IEEE 802.1D laufen lassen können. Das Programm wird einem auch die Möglichkeit bieten, verschiedene Attribute der Switches und ihrer Ports nach belieben einzustellen. Neben dem reinen Algorithmus werden die Hosts sich gegenseitig auch noch Nutzdaten zuschicken, wodurch die einzelnen SAT-Mac Tabellen aufgebaut werden. Ein weiteres Ziel ist es, die Switches mittels Threads parallel und unabhängig voneinander arbeiten zu lassen. Dies hat zur Folge, dass die Switches auf kein globales Wissen zugreifen können. Es gibt keine übergeordnete Instanz, die alle Switches lenkt und steuert. Diese Realisierung wird der echten Arbeitsweise eines Netzwerks näher kommen, als wenn alle Switches immer sofort über alle Abläufe innerhalb des Netzwerks bescheid wissen.

In this bachelor thesis a Java program is developed, which can be used to visualize networks. These networks have to be described by configuration files. They have only to consist of layer 2 switches and hosts. If you have built such a configuration file, it can be loaded by the programme, which will visualize the network. Then you can start the spanning tree algorithm IEEE 802.1D. The program offers you the possibility to change different attributes of the switches and hosts. Also the hosts will send messages to each other. Another goal is to realize the switches as threads so that they work parallel and independent of each other. Because of this the switches don’t have a global knowledge. There is no instance, which leads and controls all the switches. This implementation will be more similar to the real work of a network.

Danksagung

An dieser Stelle möchte ich meinem Betreuer Frank Bohdanowicz herzlich danken, dass er mir bei meiner Bachelorarbeit immer hilfreich zur Seite stand und mir meine Fragen schnell und verständlich beantwortet hat.

Des Weiteren möchte ich noch Daniel Janke danken, der mir ebenfalls mit Rat und Tat zur Seite stand, wenn ich mich vor einem schier unlösbaren Problem bei der Entwicklung des Programms sah. Auch hat er meine Texte auf alle möglichen Fehler hin untersucht.

Inhaltsverzeichnis

1. Einleitung 1
1.1. Vorgeschichte und Aufgabenstellung 1
1.2. Aufbau der Bachelorarbeit 1

2. Grundlagen 3
2.1. Allgemeine Beschreibung des Spanning Tree Algorithmus 3
2.2. Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D 4
2.3. Rapid-STP & Multiple-STP 15

3. Anforderungen 16
3.1. Systemgrenzen 16
3.2. Funktionale Anforderungen 16
3.3. Nicht-Funktionale Anforderungen 21

4. Das Hauptfenster 22
4.1. Menüleiste 22
4.2. Kopfbereich 23
4.3. Bereich zur Darstellung des Netzwerks 25
4.4. Bereich zur Darstellung des Spanning Tree Algorithmus 25

5. Die Nebenfenster 28
5.1. Configurations 28
5.2. SAT-Mac Tables 32
5.3. All received BPDUs 34

6. Realisierung eines Netzwerks 36
6.1. Bestandteile eines Netzwerks 36
6.2. Aufbau eines Configuration-Files 36
6.3. Visualisierung eines Netzwerks 42

7. Implementation 44
7.1. Programmstruktur 44
7.2. Implementation des Parsers 46
7.3. Implementation des Unparsers 53
7.4. Implementation der Visualisierung eines Netzwerks 56
7.5. Implementation des Hauptfensters 60
7.6. Implementation der Switches 61
7.7. Implementation der Ports 73
7.8. Implementation der Wire 74
7.9. Implementation der Hosts 74
7.10. Implementation einer BPDU 76
7.11. Implementation einer Message 76

7.12. Implementation des Startens des Algorithmus ……...…….. 76
7.13. Implementation des Pausierens des Algorithmus ……...……. 77
7.14. Implementation des Stoppens des Algorithmus ……..…… 78
7.15. Implementation der SAT-Mac Tables ……..…… 78
7.16. Implementation der Configurations ……...……79

8. Handhabung des Programms ……...……...80
8.1.Einrichten des Programms ……80
8.2.Bedienung des Programms ……...…80

9. Ausblick und Zusammenfassung ……..……. 83
9.1.Ausblick ……..…….. 83
9.2.Zusammenfassung ……...… 84

A. Inhalt der CD ……..…….. 85
B. Abbildungsverzeichnis ……..…….86
C. Tabellenverzeichnis ……87
D. Listingverzeichnis ……..…….. 88
E. Literaturverzeichnis ……...…….89

1 Einleitung

In dieser Bachelorarbeit geht es um eine Implementation des Spanning Tree Algorithmus IEEE 802.1D in Java. Um die Anschaulichkeit zu erhöhen, wurde entschieden, auch das Netzwerk, auf dem der Algorithmus laufen wird, zu visualisieren. Für eine möglichst realitätsnahe Abarbeitung sind die Switches als Threads realisiert, die parallel und unabhängig voneinander arbeiten.
In diesem einleitenden Kapitel wird erst einmal erläutert, wie es überhaupt zu dieser Bachelorarbeit gekommen ist und was die ursprüngliche Aufgabenstellung war. Im Anschluss wird ein kurzer Überblick über den Aufbau dieser Arbeit gegeben.

1.1 Vorgeschichte und Aufgabenstellung

Die Idee zu dieser Bachelorarbeit entstand hauptsächlich dadurch, dass Prof. Dr. Steigner in seiner Vorlesung „Grundlagen der Rechnernetze“ erwähnt hatte, dass seine Arbeitsgruppe ein Programm sucht, das den Spanning Tree Algorithmus simuliert. Nach einem ersten Gespräch mit Prof. Dr. Steigner und Frank Bohdanowicz stellte sich heraus, dass sie für solch ein Programm wirklich Verwendung hätten. Ziemlich schnell wurde sich darauf geeinigt, dass die ursprüngliche Form des Spanning Tree Algorithmus also die Version IEEE 802.1D mit einer anschaulichen Visualisierung des Netzwerks implementiert werden solle. Auch die Wahl der Programmiersprache stand schon direkt zu Beginn fest und zwar Java. Der Arbeitsgruppe war klar, dass dies alles zusammengenommen für eine Bachelorarbeit viel zu viel ist. Aber da man zu diesem Zeitpunkt noch nicht wusste, wie aufwendig die einzelnen Teilgebiete sein würden, entschied man sich dazu, erst im Laufe des Entstehungsprozesses des Programms Teilgebiete, die zu schwer oder zu aufwendig wären, wegzulassen. Im Laufe der nächsten Gespräche zeigte sich, dass es zu aufwendig wäre, ein Netzwerk im laufenden Programm vom Benutzer mittels Drag&Drop selbst zusammenbauen zu lassen. Als Ausweg wählte man den Weg über ein Configuration-File, mit der der Nutzer schon im Vorfeld sein Netzwerk erstellen solle. Dieser Anpassungsprozess zog sich durch die ganze Entstehungsphase des Programms und mündete schließlich in der fertigen Anforderungsliste, auf die im Kapitel 3 auf Seite 16 näher eingegangen wird. Ein besonderer Wunsch der Arbeitsgruppe war, dass die einzelnen Komponenten eines Netzwerks als Threads realisiert werden sollen, damit man die Arbeitsweise des Algorithmus möglichst realitätsnah nachbilden könne. Dies erschwerte natürlich die Umsetzung des Algorithmus, da es nun keine übergeordnete Instanz geben konnte, die alle Komponenten des Netzwerks steuerte.

1.2 Aufbau der Bachelorarbeit

Diese Bachelorarbeit gliedert sich, wie im Inhaltsverzeichnis nachzulesen ist, in acht Kapitel. Nach der Einleitung geht es nun direkt zu Kapitel 2, in dem erst einmal auf alle notwenigen Grundlagen eingegangen wird, die eine Vorraussetzung zum Verständnis dieser Arbeit darstellen. Zuerst wird allgemein beschrieben, was der Spanning Tree Algorithmus überhaupt ist, warum er entwickelt wurde und was seine Aufgabe ist. Danach wird die Entstehungsgeschichte des Spanning Tree Algorithmus zusammengefasst, um dann einen kurzen Überblick über die unterschiedlichen Variationen dieses Algorithmus zu geben. Im Hauptteil dieses Kapitels wird dann die Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D an einem Beispiel ausführlich erklärt. Dies stellt den ersten der drei großen Teile dieser Arbeit dar.

Im Anschluss folgt Kapitel 3, in dem, wie vorher schon erwähnt wurde, auf die endgültige Anforderungsliste eingegangen wird.

Im nächsten Kapitel wird dem Leser der Aufbau des Hauptfensters des Programms mit Hilfe von Screenshots erläutert. Es werden hier die drei Bereiche des Fensters mit ihren Aufgaben gezeigt, um dann in Kapitel 5 genauer auf die Realisierung eines Netzwerks einzugehen. Zuerst wird erklärt woraus sich Netzwerke, die in diesem Programm verwendet werden, zusammensetzen. Dann wird der Aufbau eines Configuration-Files näher erläutert, um am Ende zu zeigen, wie solch ein Netzwerk im Programm visualisiert wird.
Nun folgt der zweite große Teil dieser Arbeit und zwar die Implementation, die in Kapitel 6 behandelt wird. Den Anfang macht ein Klassendiagramm vom Programm, um erst einmal einen groben Überblick über dieses zu gewinnen. Im Anschluss wird die Arbeitsweise der einzelnen Komponenten eines Netzwerks beschrieben und wie diese realisiert wurden. Als Abschluss dieses Kapitels wird dann die Implementation der graphischen Benutzeroberfläche des Programms vorgestellt.
Den dritten großen Teil dieser Arbeit bildet das Kapitel 7. Hier werden an einem Beispiel alle Funktionalitäten des Programms ausführlich vorgestellt und mit vielen Screenshots veranschaulicht. Unter anderem wird gezeigt, wie man ein Configuration-File ins Programm lädt, den Spanning Tree Algorithmus startet, die Einstellungen der einzelnen Komponenten verändern kann und wo man die SAT-Mac Tabelle einsehen kann.
Im letzten Kapitel dieser Bachelorarbeit wird ein Ausblick gegeben, wie man das Programm noch erweitern könnte und als Abschluss werden dann die wichtigsten Punkte dieser Arbeit kurz zusammengefasst.

2 Grundlagen

In diesem Kapitel werden alle zum Verständnis dieser Arbeit notwendigen Grundlagen erklärt. Dies bedeutet, dass hier auf den Spanning Tree Algorithmus eingegangen wird. Im ersten Unterkapitel wird allgemein beschrieben, was der Spanning Tree Algorithmus überhaupt ist, warum er entwickelt wurde und was seine Aufgabe ist. Im nächsten findet man eine Zusammenfassung der Entstehungsgeschichte der ersten Version dieses Algorithmus also dem mit der Bezeichnung IEEE 802.1D. Im folgenden Unterkapitel wird eine Übersicht über die wichtigsten Ableger dieses Algorithmus gegeben, bevor dann im dritten und letzten Unterkapitel die Funktionsweise dieses ursprünglichen Algorithmus anhand eines Beispiels ausführlich erklärt wird.

2.1 Allgemeine Beschreibung des Spanning Tree Algorithmus

Nicht jeder, der diese Bachelorarbeit liest, weiß sofort was überhaupt der Spanning Tree Algorithmus ist und wozu er entwickelt wurde. Er dient dazu, Schleifen in einem Netzwerk aufzubrechen und wurde von Radia Perlman im Jahre 1985 entwickelt. Er wurde dann in der Norm IEEE 802.1D standardisiert. Was Schleifen in einem Netzwerk sind, kann sich wohl jeder vorstellen: Von einen Startpunkt aus gibt es mehrere Wege innerhalb des Netzes, über die man ein bestimmtes Ziel erreichen kann.
Nun dürften sich viele Fragen, was daran so schlimm sein soll. Da nicht jeder Switch eines Netzwerks immer weiß, wo das Ziel liegt, an das er die empfangene Nachricht weiterschicken soll, macht er einen Broadcast über alle seine Ports. Der Switch schickt also die Nachricht über alle seine Ports raus. Wenn man in diese Überlegung nun einmal Schleifen mit einbezieht, wird einem klar, was für eine Gefahr sie darstellen. Es kann nämlich passieren, dass eine Nachricht immer wieder im Kreis von Switch zu Switch geschickt wird und so ihr Ziel niemals erreicht. Dies ist zwar ärgerlich aber nicht wirklich gefährlich. Dann muss man seine Nachricht halt nochmals schicken. Aber das viel größere Problem ist, dass diese immer im Kreis laufende Nachricht alle anderen Nachrichten, die über dieselbe Leitung geschickt werden, behindert und ausbremst. Im schlimmsten Fall kann dies sogar zum vollständigen Erliegen jeglichen Datenverkehrs innerhalb des Netzwerks führen.
Nun erkennt wohl jeder die Problematik von Schleifen in einem Netzwerk. Schleifen haben nicht nur ausschließlich schlechte bzw. nachteilige Eigenschaften. Der größte Vorteil von Schleifen ist die Ausfallsicherheit eines Netzwerks. Sollte nämlich einmal einer der redundanten Wege ausfallen, ist immer noch ein anderer da, über den weiterhin Daten übertragen werden können. Deshalb ist es manchmal sogar erwünscht, Schleifen in einem Netzwerk zu haben.
Aber wie kann man sicherstellen, dass in einem Netzwerk, das nun einmal Schleifen enthält, die Daten trotzdem zuverlässig ihr Ziel erreichen und nicht immer im Kreis laufen? Die Lösung stellt der Spanning Tree Algorithmus dar. Er befindet sich auf jedem Layer 2 Switch innerhalb eines Netzwerks. Denn nur dort, lassen sich Schleifen wirklich unterbinden. Auf den Hosts, die die Daten verschicken und empfangen, würde solch ein Algorithmus nichts bringen. Von dort aus könnte man keine Schleifen aufbrechen. Also blieb nur die Möglichkeit, diesen Algorithmus auf den Switchen selber laufen zu lassen.
Jeder Switch kommuniziert nun mit den anderen über kleine Management-Nachrichten, den so genannten „Bridge Protocol Data Units“ oder kurz BPDUs. Mit Hilfe dieser Nachrichten bestimmen sie einen Root-Switch. Dies ist einer der Switches im Netzwerk, der zum Wurzelknoten des Spanning Trees wird. Dann bestimmen alle anderen Switches, welcher ihrer Ports zum Root-Port wird und welche Blocking- bzw. Designated-Ports werden. Durch das Blockieren redundanter Pfade werden Schleifen aufgebrochen und zuverlässiges Verschicken von Daten ermöglicht. Sollte später einmal ein Switch oder eine Leitung

ausfallen, rekonfiguriert sich der Spanning Tree selbstständig, um wieder ein Verschicken der Daten zu ermöglichen. [5, 13]

2.2 Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D

In diesem Unterkapitel wird die Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D anhand eines einfachen Beispiels ausführlich erklärt. Die Screenshots, die zur Veranschaulichung verwendet werden, zeigen alle einen Teil des Hauptfensters des Programms, das in Kapitel 4 auf Seite 22 näher erklärt wird.
Nachdem man ein Configuration-File geladen hat, erscheint sofort das visualisierte Netzwerk. Als Beispiel verwende ich hier das „Configuration File 01.xml“, das auf der CD vorhanden ist. Nun dürfte man das Netzwerk aus Abbildung 01 im rechten Panel sehen.

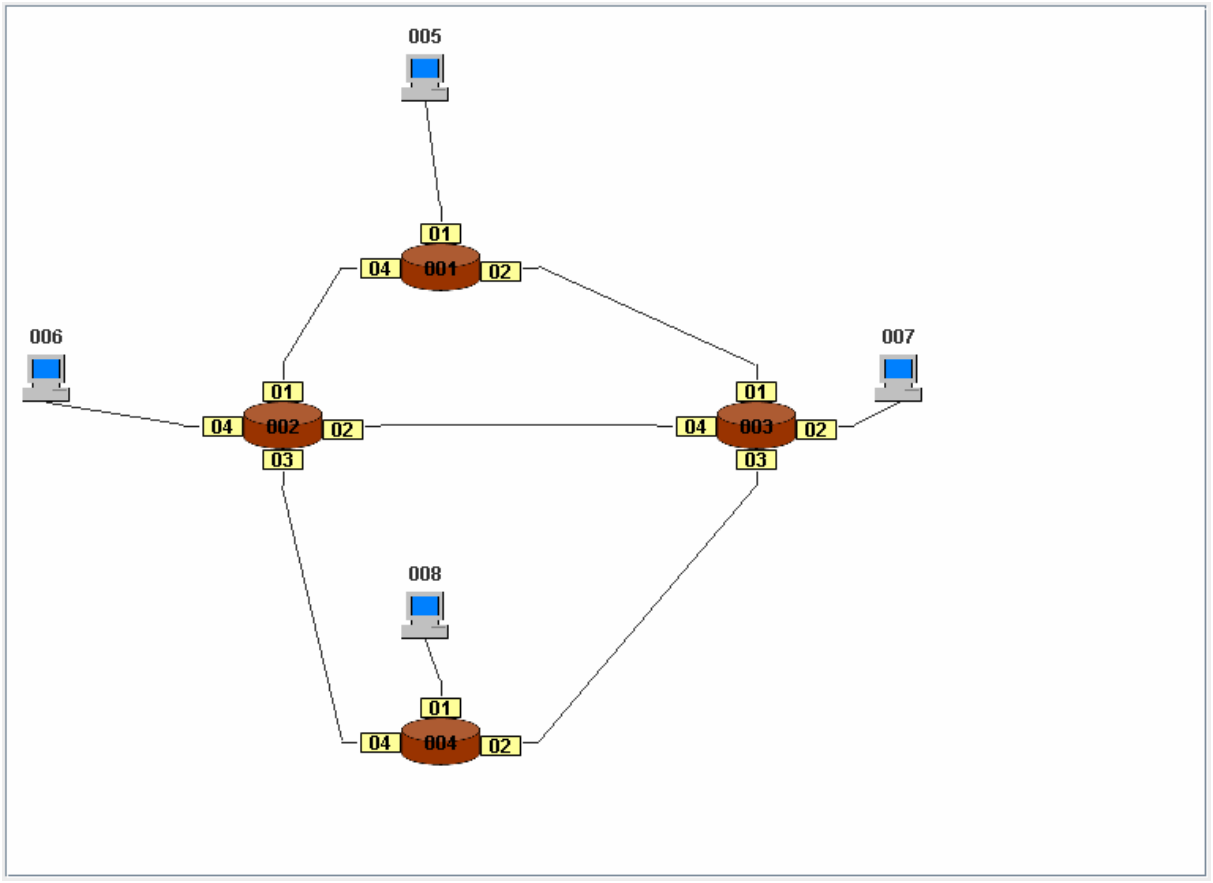

Abbildung 01: Netzwerk nach dem Laden des Configuration File 01

Wie man in Abbildung 01 sehen kann, besteht dieses Netzwerk aus vier Switches mit den IDs 001, 002, 003 und 004 und vier Host mit den IDs 005, 006, 007 und 008. Jeder der vier Switches hat drei oder vier Ports, von denen jeder eine eigene ID hat. Wie man an diesem sehr einfachen Beispiel sehen kann, gibt es drei Schleifen innerhalb dieses Netzwerks und zwar zwei kleine die aus den Switchen 001, 002 und 003 bzw. aus den Switchen 002, 003 und 004 und eine große, die aus allen vier Switchen besteht.

Bevor nun auf die Funktionsweise des Spanning Tree Algorithmus näher eingegangen wird, gibt es einige Begrifflichkeiten, die im Vorfeld geklärt werden sollten.

Beschreibung der beiden Switch-Arten:

Das Ziel des Algorithmus ist es, einen Spannbaum, den so genannten Spanning Tree, aufzubauen. Wie der Name schon nahe legt, baut der Algorithmus einen Baum auf. Jeder Baum muss genau einen Wurzelknoten haben und alle anderen Knoten müssen ein Elternelement haben. Alle Knoten, die die gleiche Entfernung zum Wurzelknoten besitzen, befinden sich in einer Ebene.
Dies trifft auch auf den Spannbaum zu. Am Ende des Algorithmus muss es genau einen Wurzelknoten geben und alle anderen Knoten müssen in Ebenen unterteilt sein. Die Knoten sind in diesem Fall, wie man sich sicherlich denken kann, die Switches. Der Wurzelknoten heißt hier Root-Switch, der in der Visualisierung mit grüner Schrift dargestellt wird und die anderen normalen Switches heißen Designated-Switches, die mit blauer Schrift dargestellt werden. Nur der Root-Switch darf neue BPDUs erzeugen. Alle anderen Switches empfangen diese und leiten sie verändert weiter. [3, 9]

Beschreibung der Port-Arten:

Wie schon vorher erwähnt, gibt es auch verschiedene Port-Arten und zwar Root-Port, Designated-Port und Blocking-Port. Jeder Switch muss genau einen Root-Port bestimmen, der ebenfalls wie der Root-Switch grün dargestellt wird. Dieser spezielle Port stellt sozusagen die kürzeste Verbindung zum Root-Switch dar. Er kann Nutzdaten und BPDUs empfangen aber nur erstere versenden.
Dann gibt es noch die beiden anderen Port-Arten. Die Designated-Ports sind die normalen Ports und werden blau dargestellt. Sie können BPDUs und Nutzdaten empfangen und versenden.
Am Schluss kommen dann noch die Blocking-Ports. Auch ihre Aufgabe ist nicht schwer zu erraten. Sie sind dafür zuständig, redundante Pfade im Netzwerk zu blockieren. Nur ihnen ist es zu verdanken, dass Schleifen aufgebrochen werden und es für jede Nachricht, die verschickt wird, immer nur genau einen Weg gibt, den sie nehmen kann. Sie werden in der Visualisierung rot dargestellt. Sie empfangen und leiten keine Nutzdaten weiter. BPDUs könne sie nur empfangen, aber nicht versenden.
Wie genau all diese Port-Arten bestimmt werden und welche Kriterien in die Entscheidung mit einfließen, wird später in diesem Unterkapitel bei der Beschreibung der Funktionalität des Spanning Tree Algorithmus erklärt. [3, 9]

Beschreibung der Port-Stati:

Neber den Port-Arten gibt es noch die Port-Stati, die jeder Port regelmäßig durchläuft. Hiervon gibt es insgesamt fünf Stück und zwar Disabled, Blocking, Listening, Learning und Forwarding. Sie werden abgesehen vom Disabled-Status im Programm nicht visualisiert. Jedoch ist dieser ein besonderer Status, der vom Algorithmus selbst nicht erreicht werden kann. Die einzige Möglichkeit, dass ein Port den Status Disabled erreicht, ist die, dass der Nutzer ihn manuell auf diesen Status setzt. Sobald ein Port disabled ist, spielt er für den Algorithmus selber keine Rolle mehr. Er ist dann abgeschaltet und macht gar nichts mehr. Deshalb wird er komplett grau dargestellt. Wenn der Nutzer ihn wieder aktivieren möchte, muss er dies auch wieder manuell tun. Alle anderen Stati können und werden vom Algorithmus selber erreicht.

Zur Veranschaulichung folgt nur erst einmal eine kleine Graphik:

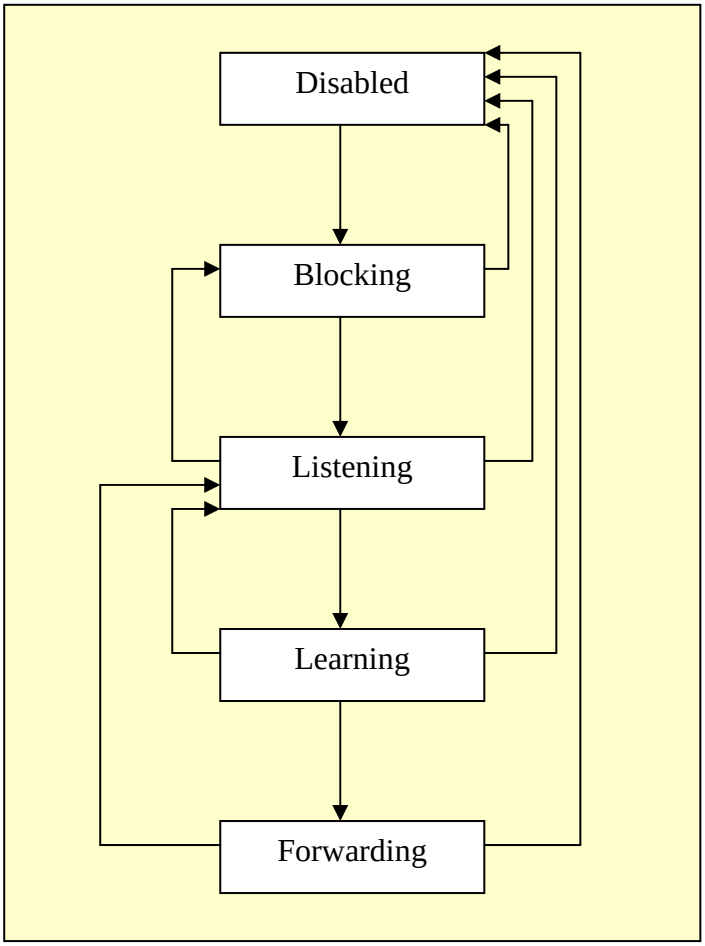

Abbildung 02: Port-Stati
Basiert auf Quelle: Schulte, Wolfgang: Spanning Tree [3]

In Abbildung 02 kann man gut erkennen, welche Port-Stati es gibt und wie sie erreicht werden können. Zuerst sind alle Ports im Blocking-Status, gehen dann über in den Listening-Status, um in den Learning-Status und am Ende in den Forwarding-Status zu gehen. Im Blocking-Status können die Ports nur BPDUs empfangen. Nutzdaten werden weder empfangen noch weitergeleitet.
Sobald eine BPDU empfangen wird, gehen sie in den Listening-Status, wo sie die Nachricht auswerten und die Switch-Arten und Port-Arten bestimmen. Kurz gesagt, bauen sie in diesem Status den Spannbaum auf. Nutzdaten können nicht empfangen und auch nicht weitergeleitet werden. In beiden Stati versenden die Ports keine BPDUs. Sollte der Port dann nicht zum Blocking-Port geworden sein, geht er in den Learning-Status. Im anderen Fall geht er sofort zurück in den Blocking-Status.
Sollte der Port den Learning-Status erreicht haben, aktualisiert er sein Wissen. Er kann zwar immer noch keine Nutzdaten empfangen oder weiterleiten, aber er kann nun zumindest BPDUs weiterleiten. Jetzt geht er in jedem Fall in den Forwarding-Status. Erst jetzt kann er Nutzdaten empfangen und weiterleiten. Ebenfalls kann er nun BPDUs empfangen und verschicken. Egal in welchen Status ein Port eine BPDU empfängt, er geht dann sofort in den Listening-Status und beginnt mit dem ganzen Ablauf von vorne. [1-12]

Anfängliches Wissen der Switches:

Jede Komponente des Netzwerks, also alle Switches, Ports und Hosts benötigen schon von Anfang an ein bestimmtes Grundwissen. Dieses muss in dem Configuration-File angegeben werden. Ein paar Beispiele für solches Grundwissen sind Mac-Adressen, IDs, Prioritäten, Pfadkosten und noch eine ganze Menge mehr. Einiges von diesem Wissen ist nur zum Aufbau des Netzwerks erforderlich, jedoch anderes benötigt der Spanning Tree Algorithmus selber. Deshalb folgt nun eine Übersicht und jeweils eine kurze Erklärung zu allen Fakten, die jeder Switch wissen muss.

	ID	**Mac-Adresse**	**Priorität**	**Max-Age**	**Hello-Time**	**Forward-Delay**
Defaultwerte	---	---	32.768	10	2	2
Werte von Switches 001	001	00:001	32.768	10	2	2
Werte von Switches 002	002	00:002	32.768	10	2	2
Werte von Switches 003	003	00:003	32.768	10	2	2
Werte von Switches 004	004	00:004	32.768	10	2	2

Tabelle 01: Anfängliches Wissen der Switches

In der ersten waagerechten Zeile sieht man die Namen der Fakten, die jedem Switch in dem Configuration-File mitgeteilt werden müssen. In der zweiten stehen die Werte zu den Fakten, die reale Switches als Defaultwerte ansehen und in den restlichen Zeilen stehen die Werte, wie sie in dem Configuration-File 01 angegeben wurden.
Was eine ID und eine Mac-Adresse ist, dürfte wohl jedem klar sein. Hier kann man auch schlecht einen Standartwert angeben. Eins sollte man aber noch zu den IDs wissen und zwar, dass sie nicht eindeutig sein müssen. Es kann also mehrere Switches mit derselben ID geben. Die Mac-Adresse hingegen muss immer eindeutig sein.
Als drittes Fakt, folgt nun die Priorität. Je niedriger die Zahl ist, desto wichtiger ist der Switch. In diesem Beispiel wurde einfach der Default-Wert beibehalten. Es wird geraten, dem Switch mit der höchsten Bandbreite die niedrigste Priorität zu geben. Damit ist gewährleistet, dass dieser Switch zum Root-Switch wird. Dies ist sinnvoll, da er den zentralen Knoten des Spannbaums darstellt. Über ihn laufen die meisten Nachrichten und er muss alle BPDUs erzeugen.
Nun kommt das Max-Age. Wie der Name schon vermuten lässt, ist hier das maximal erlaubte Alter einer BPDU angegeben. Auch hier wurde der Defaultwert bei allen Switches belassen. Die 10 steht hier jedoch nicht für Sekunden sondern für Hops. Also mit jedem Switch, den eine BPDU passiert, wird ihr Alter um 1 erhöht. Sollte ein Switch eine Nachricht erhalten, die schon 10 Hops alt ist, wird sie einfach gelöscht. Hiermit ist gewährleistet, dass eine BPDU irgendwann gelöscht wird und nicht endlos im Netzwerk umherwandert.
Die Hello-Time gibt an, in welchen Zeitabständen der Root-Switch neue BPDUs erzeugt und verschickt. Der Defaultwert ist zwei Sekunden, der auch so belassen wurde.
Als letztes Fakt folgt nun die Forward-Delay. Auch sie gibt ein Zeitintervall an. Bevor ein Port vom Listening- in den Learning-Status oder vom Learning- in den Forwarding-Status wechselt, muss er erst einmal diese hier angegebene Zeitspanne warten. Die Maßeinheit ist wieder Sekunden. Diese Wartezeiten dienen einfach dazu sicherzustellen, dass das Netzwerk nach einer Topologieänderung noch konvergent ist. [1-12]

Zusätzliches Wissen der Switches:

Nachdem nun ausführlich erklärt wurde, was für Wissen einem Switch schon in dem Configuration-File mitgeteilt werden muss, folgt nun das Wissen, dass er sich erst während dem Ablauf des Spanning Tree Algorithmus erwirbt.
Jeder Switch merkt sich nur zwei weitere Fakten und zwar die angenommenen Root-ID und die zugehörigen Root-Pfadkosten. Im ersten Fakt merkt er sich den Switch, von dem er der Meinung ist, dass er der Root-Switch sei und im zweiten die kürzesten Pfadkosten zu diesem.
Da jeder Switch nur sein zu Beginn mitgegebenes Wissen hat und nichts über die anderen angeschlossenen Komponenten weiß, kann er nur neues Wissen durch das Auswerten von empfangenen BPDUs erhalten. Dies ist ein langwieriger Prozess, aber irgendwann weiß jeder Switch im Netzwerk, wer der richtige Root-Switch ist und wie groß die Pfadkosten zu ihm sind. [3, 9]

Anfängliches Wissen der Ports:

Auch den Ports muss ein gewisses Wissen schon in dem Configuration-File mitgeteilt werden. In Tabelle 02 ist wie bei den Switches erst einmal eine Übersicht über dieses Wissen zu sehen:

Ports von Switch 001	**ID**	**Priorität**	**Pfadkosten**
Port 01	00001.01	32.768	19
Port 02	00001.02	32.768	19
Port 04	00001.04	32.768	19

Tabelle 02: Anfängliches Wissen der Ports von Switch 001

Es wird sich hier auf die Ports von Switch 001 beschränkt. Alle Ports der anderen Switches sind äquivalent aufgebaut.
Jeder Port hat wieder eine ID. Jedoch muss diese im Gegensatz zu der vom Switch eindeutig sein. Sie setzt sich aus zwei Teilen zusammen. Der erste Teil besteht aus der Mac-Adresse des Switches, zu dem der Port gehört und der zweite aus der internen ID. Jeder Switch kann zum Beispiel einen Port mit der internen ID 01 haben. Wenn die gesamte ID nur aus diesem zweiten Teil bestehen würde, wäre sie, auf das gesamte Netzwerk bezogen, nicht mehr eindeutig. Deshalb braucht man auch den ersten Teil. Jedoch spielt dieser für den Spanning Tree Algorithmus selber keine Rolle, da der Algorithmus immer nur die Ports eines Switches betrachtet. Die Priorität der Ports spielt dieselbe Rolle, wie die der Switches.
Anders als die Switches haben die Ports noch das Fakt Pfadkosten. Hier stehen die Pfadkosten der angeschlossenen Leitung drin. Nun mag sich jemand wundern, was die Zahl 19 bedeutet. Um diese Zahl zu bestimmen, gibt es eine offizielle Tabelle, wie in Tabelle 03 zu sehen.

Bandbreite	STP-Kosten
4 MBit/s	250
10 MBit/s	100
16 MBit/s	62
45 MBit/s	39
100 MBit/s	19
155 MBit/s	14
622 MBit/s	6
1 GBit/s	4
10 GBit/s	2

Tabelle 03: Zuordnung von Bandbreite und STP-Pfadkosten
Basiert auf Quelle: Schulte, Wolfgang: Spanning Tree [3]

Da die Switches in diesem Programm nur simuliert werden, kann man sich einen beliebigen Wert aussuchen. Man kann die Pfadkosten in dem Configuration-File aber auch vollkommen frei vergeben. [1-12]

Aufbau einer BPDU:

Eine BPDU oder in Langform auch „Bridge Protocol Data Unit“ genannt, ist eine Management-Nachricht, mit denen die Switches im Rahmen des Spanning Tree Algorithmus miteinander kommunizieren. Dies ist die einzige Möglichkeit für einen Switch an neues Wissen zu gelangen. Wie vorher schon erwähnt, werden sie nur vom Root-Switch erzeugt und von den anderen Switches verändert weitergeschickt. Sollte ein Switch solch eine Nachricht empfangen, geht er sofort in den Listening-Status und wertet sie aus. Allein durch die Informationen, die mittels dieser Nachrichten ausgetauscht werden, baut der Spanning Tree Algorithmus den Spannbaum auf. Also ist es sinnvoll, sich den Aufbau einer solchen BPDU einmal näher anzuschauen, bevor man direkt mit der Abarbeitung des Algorithmus beginnt. Mit Tabelle 04 wird eine kleine Übersicht über den Aufbau einer solchen Nachricht gegeben.

Protocol ID	Version	Message Type	Flags
0	0	0	0

Root ID	Root-Pathcosts	Bridge ID	Port ID
32768.001.00001	0	32768.001.00001	00001.02

Message Age	Max. Age	Hello Time	Forward Delay
1	10	2	2

Tabelle 04: Aufbau einer BPDU
Basiert auf Quelle: Schulte, Wolfgang: Pfadfinder auf Layer 2 [9]

Aus Platzgründen wurde die BPDU hier auf drei Zeilen aufgeteilt. Solch eine Nachricht besteht aus all diesen Informationen. Um die ersten drei Felder braucht man sich nicht sonderlich zu kümmern, da sie standardgemäß immer auf 0 gesetzt werden. Die erste Null bedeutet, dass der Spanning Tree Algorithmus IEEE 802.1D verwendet wird und die dritte Null sagt aus, dass es sich bei dieser Nachricht um eine BPDU handelt. Da in dieser Bachelorarbeit die Rekonfiguration des Spannbaums nicht betrachtet wird, bleibt das Flag auch immer auf 0 gesetzt.
Im nächsten Feld Root ID steht der vom sendenden Switch angenommene Root-Switch drin. Die Kennzeichnung des Root-Switches besteht aus drei Teilen, der Priorität, der ID und der Mac-Adresse des Root-Switches. Die Root-Pfadkosten sind die Pfadkosten des kürzesten

Pfades eines Switches zum angenommenen Root-Switch. In diesem Beispiel stammt die BPDU vom Root-Switch selber. Deshalb betragen die Root-Pfadkosten auch 0.
Die Bridge ID ist genauso aufgebaut wie die Root ID nur mit dem Unterschied, dass hier die Werte des sendenden Switches eingetragen werden und nicht die des Root-Switches. Daran, dass die Root ID und die Bridge ID gleich sind, kann man ebenfalls erkennen, dass diese BPDU direkt vom Root-Switch selber kommt. Jedes Mal, wenn ein Switch eine BPDU weiterleitet, trägt er hier immer seine eigenen Werte ein, da er jetzt der Sender ist.
Dasselbe gilt für die Port ID. Hier steht immer die ID des Ports drin, über den die BPDU zuletzt verschickt wurde. Wie sich die ID zusammensetzt, ist bereits in Abschnitt „Anfängliches Wissen der Ports“ erklärt worden.
In der dritten Teiltabelle folgen nun noch vier Informationen, von denen die letzten drei aus dem Abschnitt „Anfängliches Wissen der Switches“ bekannt sein dürften. Deshalb wird auf diese drei Werte nicht nochmals näher eingegangen. Nun fehlt nur noch die Message Age. Aber hier gibt der Name eigentlich schon Auskunft darüber, was sich hier verbirgt nämlich das Alter der BPDU. Bei jedem Verschicken einer BPDU wird ihr Alter um 1 erhöht und zwar bis ihr Alter das Max. Age erreicht hat. Dann wird sie einfach gelöscht. Schon beim ersten Verschicken einer BPDU wird ihr Alter auf 1 gesetzt.

Nach dieser sehr ausführlichen Vorbereitung hat man nun endlich das nötige Wissen, um den Spanning Tree Algorithmus nachvollziehen zu können. [3, 9]

Funktionalität des Spanning Tree Algorithmus:

In diesem Abschnitt folgt nun endlich die Beschreibung der Arbeitsweise des Spanning Tree Algorithmus.
Sobald man den Algorithmus startet, haben alle Switches noch keinerlei Wissen über irgendeine andere Komponente des Netzwerks. Deshalb gehen alle erst einmal davon aus, dass sie selber der Root-Switch sind. Dies erkennt man in der Abbildung 03 an der grünen Schriftfarbe der Switches. Alle Ports werden in den Blocking-Status gesetzt und werden als Designated-Ports betrachtet, was man an der blauen Schriftfarbe erkennen kann. Dies bedeutet, dass sie nur BPDUs empfangen, aber keine verschicken können.
Dies führt, wie jedem klar sein dürfte, zu einem Problem. Wie soll auch nur irgendein Switch etwas lernen, wenn keinerlei BPDUs verschickt werden. Um dies zu lösen, hat man sich dazu entschieden, alle Ports in den Forwarding-Status zu setzten. Jedoch kann man dies nicht direkt vom Blocking-Status aus tun. Wie im Abschnitt „Beschreibung der Port-Stati“ erklärt, müssen die Ports zuerst die Stati Listening und Learning durchlaufen, um endlich in den Forwarding-Status zu kommen.
Bevor vom Listening- in den Learning-Status und vom Learning- in den Forwarding-Status gewechselt werden kann, muss der jeweilige Timer erst einmal abgelaufen sein. Die Länge beider Timer ist gleich und wird über das Attribut Forward-Delay eines Switches bestimmt. Sobald alle Ports eines Switches den Forwarding-Status erreicht haben, erzeugt der Switch seine ersten BPDUs, die er über all seine Ports rausschickt. Dann wartet er die Zeit, die in der Hello Time angegeben ist und erzeugt die nächsten BPDUs. Solange er Root-Switch bleibt, erzeugt und verschickt er nach Ablauf des Zeitintervalls einfach immer neue BPDUs. Denn man erinnere sich, dass nur der Root-Switch BPDUs erzeugen darf. Alle andern Switches verändern diese nur und leiten sie weiter. Wie so eine BPDU aufgebaut ist, wurde ja im Vorfeld ausführlich beschrieben. Alle Ports eines Root-Switch bleiben Designated-Ports.

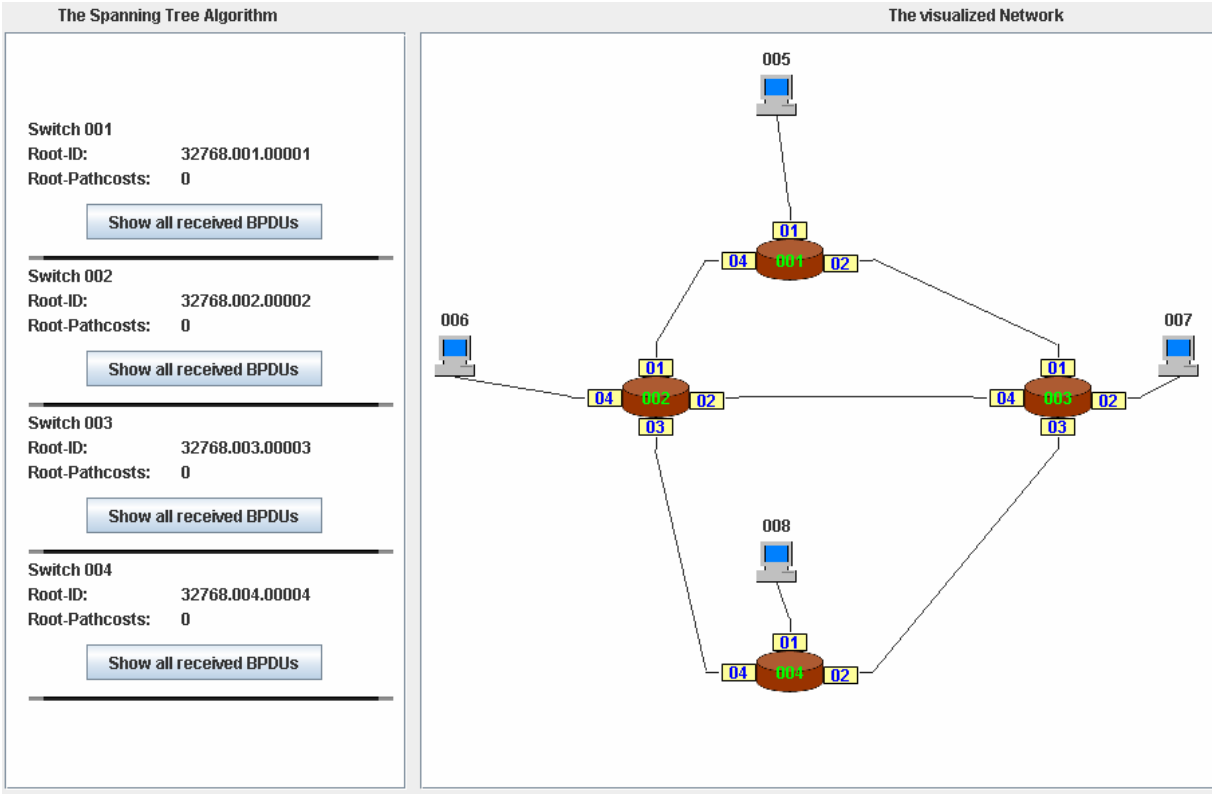

Abbildung 03: Netzwerk nach dem Starten des Algorithmus

In Abbildung 03 kann man links neber dem visualisierten Netzwerk auch das aktuelle Wissen jedes Switches erkennen. Hier sieht man, dass jeder Switch, wie eben beschrieben, zu Beginn davon ausgeht, dass er selber der Root-Switch ist.

Nun wurden also die ersten BPDUs verschickt und erreichen nach einiger Zeit den angeschlossenen Switch. Dieser merkt, dass eine BPDU angekommen ist und geht sofort in den Listening-Status. Nun wertet er diese Nachricht aus.
Als erstes wird der Root-Switch bestimmt. Dazu vergleicht er seine intern gespeicherte Root ID mit der, die in der BPDU mitgeschickt wurde. Wie vorher schon erklärt wurde, setzt sich die Root ID aus drei Teilen zusammen und zwar der Priorität, der ID und der Mac-Adresse des angenommenen Root-Switches. Genau in der gerade aufgezählten Reihenfolge werden die drei Teile der beiden Root IDs miteinander verglichen. Gewinnen tut immer der kleinere Wert. Bis zum Ende dieses Absatzes wird nur die empfangene mit der eigenen angenommenen Root ID verglichen. Ist zum Beispiel die empfangene Priorität kleiner als die eigene angenommene, erkennt der auswertende Switch, dass sein angenommener Root-Switch falsch ist. Sollten jedoch beide Prioritäten gleich sein, geht es mit der ID und dann unter Umständen sogar mit der Mac-Adresse weiter. Spätestens jetzt steht fest, ob der angenommene Root-Switch des auswertenden Switches stimmt oder nicht.
Da beim ersten Empfang einer BPDU der auswertende Switch ja selber davon ausgeht, dass er Root-Switch ist, wird hier darüber entschieden, ob er Root-Switch bleibt oder nicht. Wenn sich herausstellt, dass er es nicht mehr ist, ändert sich die Schriftfarbe auf blau, da er jetzt ein normaler Designated-Switch ist.
Als zweites folgt nun die Bestimmung des Root-Ports. Dies ist beim ersten Mal ganz einfach. Sollte der Switch erkannt haben, dass er selber kein Root-Switch mehr ist, macht er einfach den Port, über den er die BPDU empfangen hat, zum Root-Port und färbt ihn grün ein.
Nach dem Auswerten der ersten BPDU hat sich rausgestellt, dass der Switch entweder weiterhin Root-Switch ist oder falls dies nicht mehr der Fall sein sollte, hat der nun

Designated-Switch einen Root-Port bestimmt. Zu diesem Zeitpunkt ist es selbstverständlich noch nicht möglich einen Blocking-Port zu bestimmen.
Nach Ablauf des Timers geht der Port dann in den Learning-Status, wo er sich falls notwendig die neue Root ID merkt und die neuen Root-Pfadkosten berechnet. Dazu addiert er einfach zu den empfangenen Root-Pfadkosten, die ja hier noch 0 waren, die Pfadkosten der Leitung, über die die BPDU geschickt wurde, hinzu. Dann startet der zweite Timer.
Sollte auch dieser abgelaufen sein, geht der Port in den Forwarding-Status. Nun prüft der Switch, ob die BPDU von einem Switch, der eine Ebenen näher am Root-Switch liegt, gekommen ist und passt dann die empfangene BPDU gegebenenfalls an, indem er überall seine eigenen Werte einträgt. Danach schickt er sie über alle Designated-Ports, die im Learning- oder Forwarding-Status sind, raus. Dabei lässt er den Port, über den er die BPDU empfangen hat, aus. Dies alles macht jeder einzelne Switch im gesamten Netzwerk. Nun ist der Algorithmus genau einmal durchgelaufen.

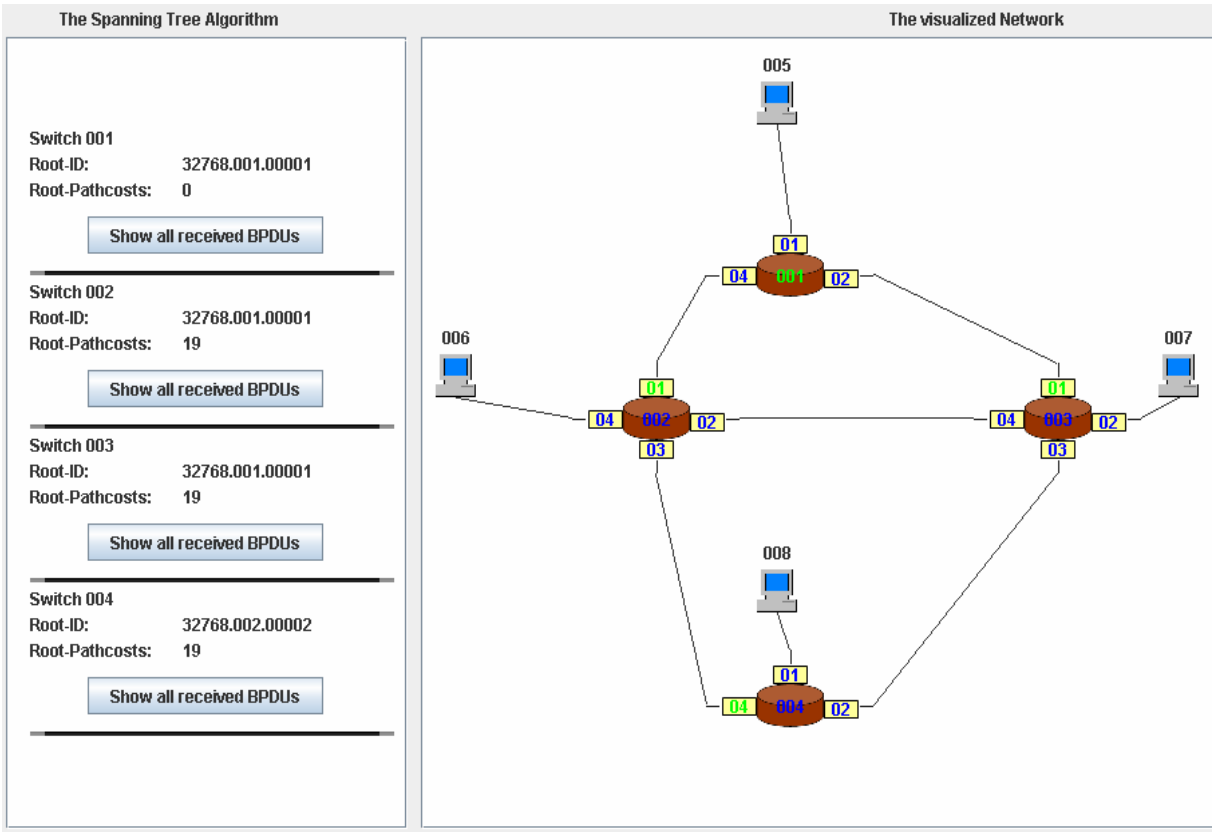

Abbildung 04: Netzwerk nach dem Auswerten der ersten BPDU

In Abbildung 04 kann man erkennen, dass jeder Switch schon eine BPDU empfangen und ausgewertet hat. Man sieht, dass bis auf Switch 001 alle davon ausgehen, dass sie nicht mehr der Root-Switch sind. Sie haben auch alle, bis der Root-Switch natürlich einen Root-Port bestimmt. Um diesen ersten Schritt richtig zu verstehen, sollte sich jeder am Besten nochmals die Tabelle 01 auf Seite 7 anschauen und den Algorithmus noch mal mit richtigen Werten durchspielen. Wenn man sich die bisher relevanten Spalten anschaut, erkennt man sofort, dass Switch 001 wirklich der Root-Switch ist. Switch 004 hat jedoch noch nicht den richtigen Root-Switch gefunden. Da er von Switch 002 seine erste BPDU empfangen hat, geht er noch davon aus, dass dieser der Root-Switch ist.
Vielleicht wird dem ein oder anderen Leser nun dir Frage in den Sinn kommen, wie Switch 004 in Erfahrung bringen kann, dass Switch 001 der Root-Switch ist, da er ja bisher noch gar keine BPDU mit diesem Wissen empfangen hat. Wenn man sich dazu den linken Bereich des

Screenshots anschaut, sieht man, dass er das noch gar nicht weiß. Er glaubt nämlich noch, dass Switch 002 der Root-Switch ist. Bevor er überhaupt von Switch 001 erfahren kann, muss er die von Switch 002 oder Switch 003 veränderte BPDU empfangen, was er zu diesem Zeitpunkt aber noch nicht getan hat. Auch sieht man im linken Bereich gut, wie die Switches ihr Wissen über die Root-Pfadkosten angepasst haben.
Nun kommt der zweiten Durchlauf des Algorithmus. Die Switches werden irgendwann die veränderten BPDUs empfangen und wie vorher auch sofort wieder in den Listening-Status gehen. Auch jetzt wird wieder zuerst die Root ID überprüft. Da Switch 002 und 003 aber schon den richtigen Root-Switch kennen, ändert sich hier nichts. Jedoch Switch 004 weiß nun endlich auch, wer der richtige Root-Switch ist.
Nun folgt wie üblich die Bestimmung des Root-Ports. Immer wenn ein Switch einen neuen Root-Switch erkennt, ernennt er sofort den Port, über den die BPDU empfangen wurde, zum neuen Root-Port. Der alte wird wieder zum Designated-Port.
Es kann aber der Fall auftreten, dass ein Switch zwar schon den richtigen Root-Switch weiß und dementsprechend auch seinen Root-Port bestimmt hat, aber er nun eine BPDU über einen Port empfängt, bei dem der Weg zum Root-Switch viel kürzer ist, als der des bisherigen Root-Ports. Also ernennt er selbstverständlich den Port zum Root-Port, der die kürzesten Root-Pfadkosten zum aktuellen Root-Switch aufweisen kann. Sollten aber auch die Root-Pfadkosten von zwei Ports gleich sein, muss der Switch die in Frage kommenden Ports genauer anschauen. Dazu vergleicht er zuerst ihre Prioritäten und sollten diese zur eindeutigen Bestimmung immer noch nicht ausreichen, zieht er auch noch ihre ID zur Entscheidung mit hinzu. Hier gelten dieselben Bedingungen wie beim Switch und zwar, dass der kleinere Wert gewinnt und dieser Port dann zum Root-Port wird. Nach dieser langwierigen Prüfung ist nun der Root-Port bestimmt.

Sollte nun eine BPDU über einen Port empfangen werden, der kein Root-Port ist und auch durch die Auswertung dieser Nachricht nicht dazu wird, muss entschieden werden, ob dieser Port ein Designated-Port bleibt oder zum Blocking-Port wird. Dazu gibt es erst einmal ein paar allgemeine Kriterien:

- Der Port, der sich gegenüber einem Root-Port befindet, muss ein Designated-Port bleiben. Schaut man sich hierzu nochmals Abbildung 04 an, betrifft dies Port 03 von Switch 003. Dieser Port muss ein Designated-Port bleiben, da Port 04 von Switch 002 ein Root-Port ist.
- Die Ports eines Root-Switches bleiben auch Designated-Ports, wie man an den beiden Ports 02 und 04 von Switch 001 erkennen kann.
- Der Port, der gegenüber einem Blocking-Port liegt, muss auch Designated-Port bleiben.

Um nun einen Blocking-Port zu bestimmen, muss man prüfen, ob die eigenen Root-Pfadkosten größer sind, als die, die in der BPDU mitgeschickt wurden, vorausgesetzt der Root-Switch ist der gleiche. Wenn die vom auswertenden Switch größer sind, muss er seinen Port zum Blocking-Port machen. Aber auch hier kann es vorkommen, dass die beiden Root-Pfadkosten gleich groß sind. Sollte dies einmal der Fall sein, muss man die Bridge ID als Entscheidungskriterium nehmen. Wie vorher schon erklärt, setzt sie sich ebenfalls wie die Root ID aus denselben drei Teilen zusammen. Nun muss man diese drei Teile mit den entsprechenden Werten des auswertenden Switches vergleichen. Hier gilt anders als vorher, dass der größere Wert gewinnt. Also der Switch, der die größere Priorität, die größere ID oder unter Umständen die größere Mac-Adresse hat, muss seinen Port zum Blocking-Port machen.
Sobald ein Port zum Blocking-Port wird, geht er sofort in den Blocking-Status und wird in der Visualisierung des Netzwerks mit roter Schrift dargestellt.
Sollte der Port ein Designated-Port bleiben, geht er, wie vorher schon beschrieben, in den Learning- und dann in den Forwarding-Status. Sollte die BPDU von einem Switch kommen,

der eine Ebene näher am Root-Switch liegt, passt er die BPDU wieder an und schickt sie über all seine anderen Designated-Ports raus.

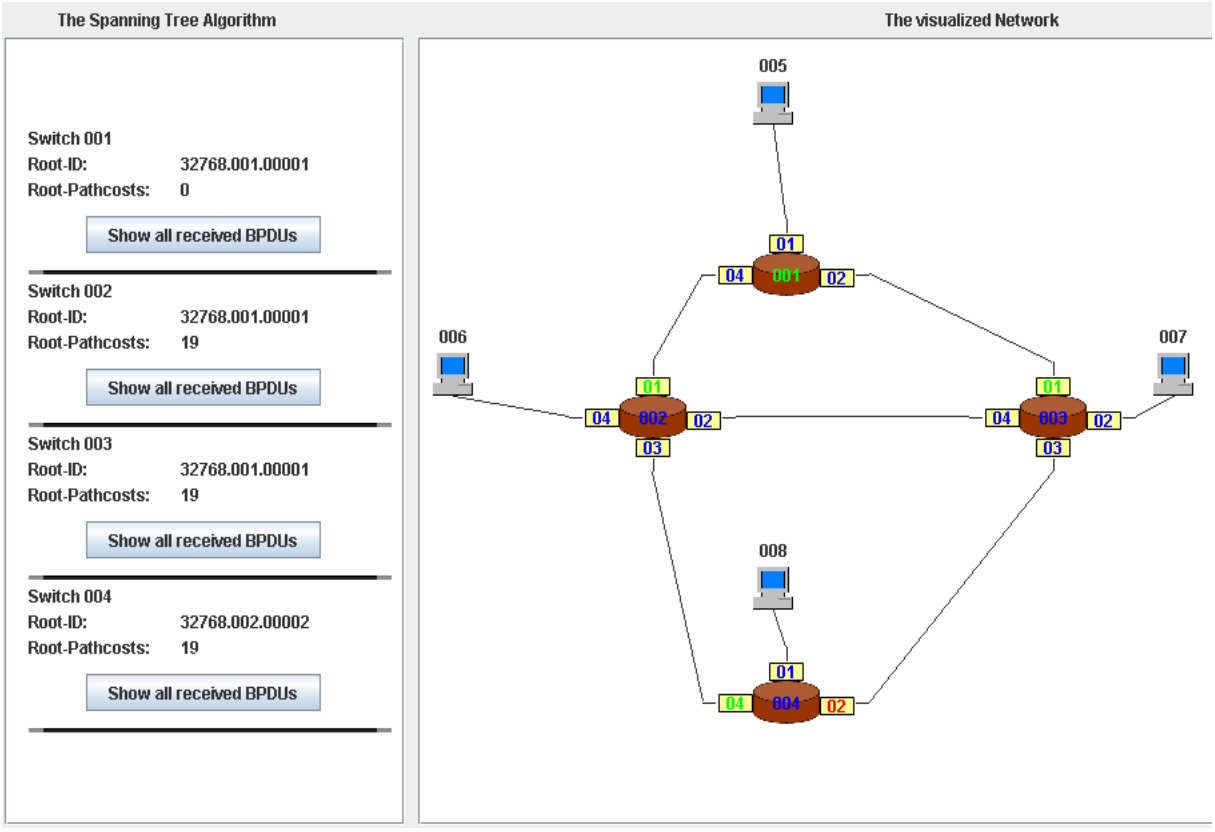

Abbildung 05: Netzwerk nach dem Bestimmen des ersten Blocking-Ports

In Abbildung 05 kann man sehen, wie der Port 02 von Switch 004 zum Blocking-Port ernannt wurde. Wenn man sich jedoch den linken Bereich etwas genauer anschaut, erkennt man, dass dieser Switch noch immer davon ausgeht, dass Switch 002 der Root-Switch ist. Also ist der Algorithmus noch nicht fertig.
Da es immer genau einen Root-Switch in einem Netzwerk geben muss, werden also, je nach eingestellter Hello Time, immer neue BPDUs erzeugt und verschickt. Diese werden dann von den Switchen ausgewertet und weitergeleitet. Irgendwann weiß jeder Switch im Netzwerk, wer der richtige Root-Switch ist, was der richtige Root-Port ist und welche Ports geblockt werden müssen. Dies kann aber je nach Größe des Netzwerks einige Zeit in Anspruch nehmen.

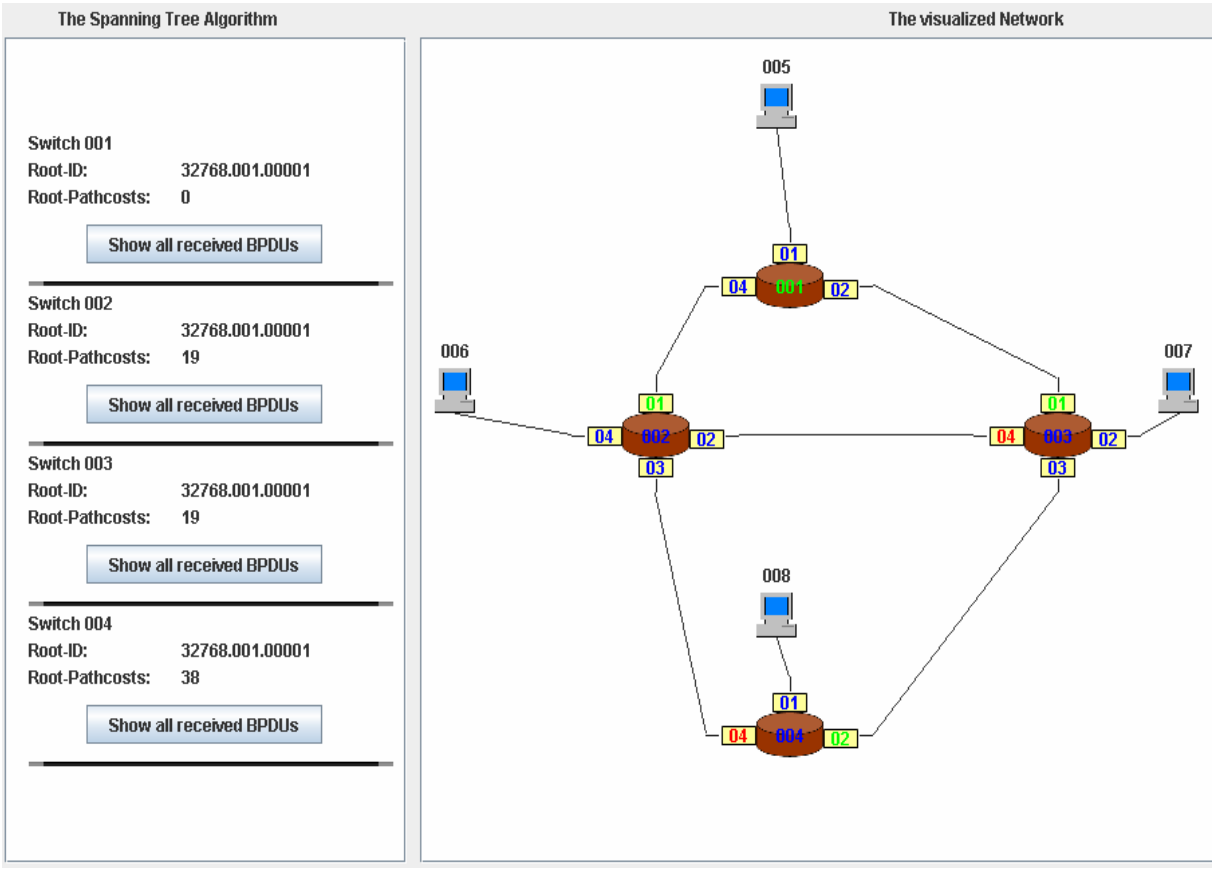

Abbildung 06: Netzwerk nach Beendigung des Algorithmus

In Abbildung 06 sieht man nun den fertigen Spannbaum mit allen Blocking-Ports. Auch im linken Bereich erkennt man, dass jeder Switch endlich den richtigen Root-Switch und die richtigen Root-Pfadkosten kennt. [1-12]

2.3 Rapid-STP & Multiple-STP

Der Rapid Spanning Tree Algorithmus wurde 1998 entwickelt und wurde unter der Norm IEEE 802.1W standardisiert. Er ist der Nachfolger des normalen Spanning Tree Algorithmus von 1985. Der große Vorteil vom neuen Algorithmus ist, dass bei einer Topologieänderung nicht sofort der gesamte Spanning Tree gelöscht und anschließend mühsam wieder neu aufgebaut werden muss. Dies beschleunigt das Rekonfigurieren des Spanning Trees enorm. [5, 13]

Der Multiple Spanning Tree Algorithmus wurde 2003 entwickelt und zuerst unter der Norm IEEE 802.1S standardisiert. Später wurde diese Norm jedoch zu IEEE 802.1Q geändert. Dieser Algorithmus ermöglicht das Verwenden von VLans in Netzwerken, da jedes VLan einen eigenen Spanning Tree aufbaut. [5, 13]

3 Anforderungen

In diesem Kapitel geht es um die Anforderungen, die dem Programm dieser Bachelorarbeit zu Grunde liegen. Sie wurden während dem Entstehungsprozess fortwährend aktualisiert.
Zuerst werden die Systemgrenzen verdeutlicht, damit jedem direkt klar ist, was nicht mehr Bestandteil des Programms ist und vor allem für welche Plattformen es ausgelegt ist. Anschließend findet man eine Liste aller funktionalen Anforderungen, die in zahlreiche Abschnitte untergliedert wurden. Zum Schluss folgen dann noch die nicht funktionalen Anforderungen, die sich hauptsächlich mit der Visualisierung und der Sprache befassen.

3.1 Systemgrenzen

In dem Unterkapitel „Systemgrenzen“ werden die Grenzen von dem Programm festgesetzt, das im Rahmen dieser Bachelorarbeit erstellt wird. Es wird klar definiert, für welche Zielplattformen und Anwendungsfälle das Programm entwickelt wird.

1. Als Zielplattform wird Windows XP, Windows Vista und Linux Open Suse 11.1 verwendet.
2. Es kann immer nur ein Netz zur selben Zeit simuliert werden.
3. Die Rekonfiguration des Spanning Trees ist nicht Bestandteil dieser Bachelorarbeit.

3.2 Funktionale Anforderungen

In dem Kapitel „Funktionale Anforderungen“ werden alle Anforderungen erhoben, die sich direkt mit der Funktionalität des Programms befassen. Das bedeutet, dass man hier nur Anforderungen findet, die beschreiben, was das Programm leisten können muss. Auf zum Beispiel gestalterische Aspekte und die Sprache wird hier nicht eingegangen.

3.2.1 Programmmanagement

Im Unterkapitel „Programmmanagement“ werden alle Anforderungen aufgeführt, die die Steuerung des Programms festlegen. Dies sind z.B. das Laden von Configuration-Files in das Programm, das Speichern dieser und das Beenden des gesamten Programms. All diese Punkte werden in einem entsprechenden Menü wieder zu finden sein.

3.2.1.1 Netzwerk laden

Das Unterkapitel „Netzwerk laden“ enthält Anforderungen, die sich mit dem Laden von Konfigurations-Dateien und dem anschließenden Anzeigen des entsprechenden Netzwerks befassen.

4. Es muss den Menüpunkt „Load Network“ geben. [Pflicht]
5. Nach Betätigung des Menüpunkts „Load Network“ muss ein Fenster erscheinen, wo man das gewünschte Configuration-File auswählen kann. [Pflicht]
6. Die in dem File definierten Komponenten, Verbindungsleitungen und Konfigurationen müssen richtig und vollständig erkannt und ausgelesen werden. [Pflicht]
7. Die Positionen der einzelnen Komponenten und Verbindungsleitungen muss ebenfalls ausgelesen bzw. ermittelt werden. [Pflicht]
8. Es muss für die Positionen der einzelnen Komponenten Default-Werte geben. [Pflicht]
9. Das Netzwerk muss im Bereich des Programms angezeigt werden, der für die Visualisierung des Netzwerkes zuständig ist. [Pflicht]

10. Sollte zum Zeitpunkt der Betätigung des Menüpunkts „Load Network“ schon ein Netzwerk im Programm geladen sein, muss eine Kontrollfrage erscheinen, ob man wirklich fortfahren möchte. [Pflicht]
11. Bei positiver Beantwortung muss die Darstellung des alten Netzwerks sowie all ihre Werte aus dem Programm entfernt werden. [Pflicht]
12. Bei negativer Beantwortung muss das bereits angezeigte Netzwerk mit seinen Werten unverändert bestehen bleiben. [Pflicht]
13. Tritt beim Laden ein Fehler auf, muss eine entsprechende Fehlermeldung angezeigt werden. [Pflicht]

3.2.1.2 Netzwerk speichern

Hier werden alle Anforderungen zu finden sein, die sich mit dem Speichern des Netzwerks befassen. Unter anderem wird hier festgelegt, dass die beim Speichern erzeugte Datei ein XML-File sein muss und sie sich auch wieder korrekt ins Programm laden lassen muss.

14. Es muss den Menüpunkt „Save Network“ geben. [Pflicht]
15. Nach Betätigung des Menüpunkts „Save Network“ muss ein Fenster erscheinen, wo man sich den gewünschten Zielort aussuchen kann. [Pflicht]
16. Man muss in der Lage sein, der zu speichernden Datei einen Namen zu geben. [Pflicht]
17. Die zu speichernde Datei sollte schon einen Default-Namen haben. [Optional]
18. Wenn man mit OK bestätigt, muss an dem angegebenen Ort ein XML-File mit dem angegebenen Namen erzeugt werden. [Pflicht]
19. Dieses erzeugte XML-File muss wieder ins Programm geladen werden können. [Pflicht]
20. In dieser Datei müssen alle Komponenten des Netzwerks mit all ihren Verbindungsleitungen vorhanden sein. [Pflicht]
21. Die Reihenfolge der Komponenten in der neu erzeugten Datei sollte dieselbe sein wie die beim Parsen der Datei. [Optional]
22. Die aktuellen Werte aller Attribute der einzelnen Komponenten müssen in der Datei enthalten sein. [Pflicht]
23. Auch die aktuellen Positionen der einzelnen Komponenten müssen gespeichert werden. [Pflicht]
24. Es sollte nur möglich sein das Netzwerk zu speichern, wenn der Algorithmus nicht am laufen ist. [Pflicht]
25. Tritt beim Laden ein Fehler auf, muss eine entsprechende Fehlermeldung angezeigt werden. [Pflicht]

3.2.1.3 Hilfe aufrufen

In dem Unterkapitel „Hilfe aufrufen“ sind ausschließlich optionale Anforderungen enthalten, die sich alle auf mögliche Hilfe-Funktionalitäten des Programms beziehen. Dies könnte z.B. ein Handbuch sein, das die Bedienung erklärt.

26. Es sollte den Menüpunkt „Help“ geben. [Optional]
27. Bei Betätigung des Menüpunkts „Help“ sollte das Handbuch geöffnet werden. [Optional]
28. Das Handbuch sollte in deutscher Sprache und als PDF-Dokument vorliegen. [Optional]

29. Sollte ein Fehler auftreten oder die erforderliche Datei nicht vorhanden sein, sollte eine entsprechende Fehlermeldung geworfen werden. [Optional]

3.2.1.4 Programm beenden

Das Unterkapitel „Programm beenden“ befasst sich ausschließlich, mit dem Thema des Beendens des laufenden Programms. Es wird hier z.B. festgelegt, ob eine Kontrollfrage beim Beenden von Nöten ist.

30. Es muss den Menüpunkt „Exit“ geben. [Pflicht]
31. Nach Betätigung des Menüpunkts „Exit“ muss eine Kontrollfrage erscheinen, ob man das Programm wirklich beenden möchte. [Pflicht]
32. Sollte diese Kontrollfrage positiv beantwortet werden, muss das Programm beendet werden. [Pflicht]
33. Beim Beenden des Programms muss die Simulation beendet und alle Fenster geschlossen werden. [Pflicht]
34. Bei negativer Beantwortung der Kontrollfrage muss das Programm unverändert bestehen bleiben. [Pflicht]

3.2.2 Programmstart

Im Unterkapitel „Programmstart“ werden alle Anforderungen aufgeführt, die sich auf die notwendigen Arbeitsschritte beim Starten des Programms beziehen. In diesem Fall wird nur das Hauptfenster des Programms aufgerufen.

3.2.2.1 Aufruf des Hauptfensters

Wie bereits vorher erwähnt, muss beim Programmstart das Hauptfenster aufgerufen werden. Nach erfolgreichem Erscheinen müssen einige Bedingungen erfüllt sein, die hier im Unterkapitel „Aufruf des Hauptfensters“ näher definiert werden. Es muss festgelegt werden, was beim Programmstart im Hauptfenster angezeigt werden soll.

35. Beim Starten des Programms muss das Hauptfenster erscheinen. [Pflicht]
36. Der Bereich „The visualized Network“, der für die Visualisierung der Netzwerke zuständig ist, muss leer sein. [Pflicht]
37. Der Bereich „The Spanning Tree Algorithm“, der das aktuelle Wissen der Switches während des laufenden Algorithmus anzeigt, muss komplett leer sein. [Pflicht]
38. Alle drei Buttons zum Starten, Pausieren und Beenden des Algorithmus müssen sichtbar aber nicht anklickbar sein. [Pflicht]
39. Es dürfen nur die Menüpunkte „Load Network“, Exit und Manual anklickbar sein. [Pflicht]
40. Sollte beim Starten des Programms ein Fehler auftreten, muss eine entsprechende Fehlermeldung erscheinen. [Pflicht]

3.2.3 Netzwerk erstellen

Die Anforderungen im Unterkapitel „Netzwerk erstellen“ befassen sich alle mit dem Erstellen von Netzwerken mittels eines Configuration-Files. Der Anwender des Programms muss im Vorfeld innerhalb einer solchen Datei sein gewünschtes Netzwerk aufbauen und diese dann ins Programm laden können.

3.2.3.1 Configuration-File erstellen

Das Unterkapitel „Configuration-File erstellen“ enthält Anforderungen, die sich alle mit dem Erstellen von Netzwerken mittels einer solchen Datei befassen. Es wird hier festgelegt, welche Komponenten es geben darf, wie man diese verbinden und konfigurieren kann.

41. Das Netzwerk muss außerhalb des Programms in einem separaten Configuration-File erstellt werden. [Pflicht]
42. Diese Datei muss in XML erstellt werden. [Pflicht]
43. Es muss für jedes Configuration-File eine einheitliche Struktur geben. [Pflicht]
44. Es muss die Komponenten Switches und Hosts geben. [Pflicht]
45. In der Konfigurationsdatei müssen alle Komponenten des Netzwerks angelegt werden. [Pflicht]
46. Jede Switch- bzw. Hostkomponente muss mit denselben Attributen versehen werden. [Pflicht]
47. Jeder Switch muss die folgenden Attribute haben:
    a. ID [Pflicht]
    b. Einzigartige Mac-Adresse [Pflicht]
    c. Priorität [Pflicht]
    d. Port-Number [Pflicht]
    e. Max. Age [Pflicht]
    f. Hello Time[Pflicht]
    g. Forward Delay [Pflicht]
    h. Position X [Optional]
    i. Position Y [Optional]
48. Jeder Port muss folgende Attribute haben:
    a. Einzigartige ID [Pflicht]
    b. Priorität [Pflicht]
    c. Connected [Pflicht]
    d. Pfadkosten [Pflicht]
49. Jeder Host muss folgende Attribute haben:
    a. ID [Pflicht]
    b. Einzigartige Mac-Adresse [Pflicht]
    c. Position X [Optional]
    d. Position Y [Optional]
50. Es müssen immer genau zwei Komponenten mit einer Leitung verbunden werden. [Pflicht]
51. Switche dürfen maximal nur genau so viele Leitungen haben, wie sie Ports besitzen. [Pflicht]

3.2.4 Simulation des Spanning Tree Algorithmus

Die Anforderungen im Unterkapitel „Simulation des Spanning Tree Algorithmus“ befassen sich alle mit dem Starten und dem Ablauf der Simulation des Spanning Tree Algorithmus. Des Weiteren wird hier festgelegt, wie die BPDUs aufgebaut sind.

3.2.4.1 Aufbau der BPDUs

Das Unterkapitel „Aufbau der BPDUs“ befasst sich mit dem Aufbau der Configuration BPDUs.

52. Jede „Configuration BPDU“ muss aus folgenden Feldern bestehen [Pflicht]:
    a. Protocol-ID
    b. Protocol Version ID
    c. BPDU Type
    d. Flags
    e. Root-ID
    f. Root Path Cost
    g. Bridge-ID
    h. Port-ID
    i. Message Age
    j. Maximum Age
    k. Hello Time
    l. Forward Delay

53. Die ersten drei Felder „Protocol-ID“, „Version“ und „Message Type“ müssen konstant auf 0 gesetzt sein. [Pflicht]
54. Die Message Age muss immer kleiner gleich der Maximum Age sein. [Pflicht]
55. Die BPDUs müssen immer von einer Komponente zur nächsten geschickt werden. [Pflicht]

3.2.4.2 Start der Simulation

Die Anforderungen im Unterkapitel „Start der Simulation“ beschreiben den groben Ablauf des Spanning Tree Algorithmus. Es wird hierbei auf die verschiedenen notwendigen Stati und Aufgabenbereiche der Ports eingegangen, sowie auf die Möglichkeit die Simulation zu beenden oder zu pausieren.

56. Es muss die folgenden Portstati geben:
    a. Blocking [Pflicht]
    b. Listening [Pflicht]
    c. Learning [Pflicht]
    d. Forwarding [Pflicht]
    e. Disabled [Pflicht]
57. Es muss die folgenden Portarten geben:
    a. Blocking-Port [Pflicht]
    b. Designated-Port [Pflicht]
    c. Root-Port [Pflicht]
58. Zu Beginn müssen alle Ports den Status Blocking haben. [Pflicht]
59. Jeder Knoten muss zuerst davon ausgehen, dass er selbst der Root-Knoten ist. [Pflicht]

60. Es müssen zwischen allen Switches Configuration BPDUs ausgetauscht werden. [Pflicht]
61. Es muss der Root-Switch bestimmt werden. [Pflicht]
62. Es darf immer nur genau einen Root-Knoten geben. [Pflicht]
63. Alle anderen Switches müssen ihren Root-Port bestimmen. [Pflicht]
64. Jeder Switch darf immer nur genau einen Root-Port haben. [Pflicht]
65. Es müssen die Blocking- und Designated-Ports bestimmt werden. [Pflicht]
66. Während der Spanning Tree aufgebaut wird, dürfen keine Nutzdaten von den Switches weitergeleitet werden. [Pflicht]
67. Der Root-Switch muss in regelmäßigen Zeitabständen selbstständig BPDUs erzeugen und aussenden. [Pflicht]
68. Switches, die eine BPDU von einem in der Topologie des Spanning Trees übergeordneten Switch empfangen, müssen anschließend ebenfalls BPDUs aussenden. [Pflicht]
69. Es darf, nachdem der Algorithmus seinen Spannbaum aufgebaut hat, zu jedem Switch immer nur genau einen aktiven Weg geben. [Pflicht]
70. Es muss möglich sein, die Simulation mittels des Stop-Buttons manuell zu beenden. [Pflicht]
71. Es sollte möglich sein, die Simulation mittels Pause-Button zu pausieren. [Optional]

3.3 Nicht funktionale Anforderungen

Im Unterkapitel „Nicht Funktionale Anforderungen“ werden alle Anforderungen aufgelistet, die sich nicht direkt auf die Funktionalität des Programms beziehen. Dazu gehört beispielsweise die graphische Umsetzung des Programms, die verwendete Sprache und einige weiter Aspekte.

72. Die Komponenten Switches und Hosts müssen jeweils durch ein entsprechendes Bild repräsentiert werden. [Pflicht]
73. Bei beiden Komponenten-Arten müssen ihre jeweiligen IDs angezeigt werden. [Pflicht]
74. Die Ports der Switches müssen mit ihrer Portnummer dargestellt werden. [Pflicht]
75. Die Verbindungsleitung zwischen zwei Komponenten muss durch einen Linie repräsentiert werden. [Pflicht]
76. Der Quelltext muss kommentiert werden. [Pflicht]
77. Das Programm muss mit JavaDoc dokumentiert werden. [Pflicht]
78. Die Javakonventionen von Sun zur Entwicklung von Programmen müssen eingehalten werden. [Pflicht]
79. Das Programm muss erfolgreich gestestet werden. [Pflicht]
80. Die Benutzeroberfläche des Programms sollte in englischer Sprache geschrieben werden.[Optional]

4 Aufbau des Hauptfensters

In Kapitel 4 wird das Hauptfenster des Programms mit all seinen Bestandteilen gezeigt und erklärt. Es setzt sich aus vier Bereichen zusammen und zwar der Menüleiste, dem Kopfbereich, dem Bereich zur Darstellung des Netzwerks und zu guter letzt dem Bereich zu Darstellung des Spanning Tree Algorithmus.

4.1 Menüleiste

In diesem Unterkapitel wird die Menüleiste des Hauptfensters vorgestellt. Es wird hierbei kurz auf alle Funktionalitäten eingegangen, die man über diese Leiste aufrufen kann. Eine Beschreibung aller Funktionalitäten kommt erst in Kapitel 7 auf Seite 44

Die Menüleiste untergliedert sich in drei Oberpunkte und zwar „File“, „Run“ und „Help“. Sie ist ganz bewusst auf Englisch gehalten, damit auch Nutzer, die nicht der deutschen Sprache mächtig sind, dieses Programm bedienen können.

Wie in anderen Programmen üblich verbergen sich hinter dem Oberpunkt „File“ alle Funktionalitäten, die zur Steuerung des grundlegenden Programms notwenig sind. Dies sind „Load Network“, „Save Network“ und „Exit“. Diese drei Menüpunkte sind eigentlich selbsterklärend, aber sie werden hier trotzdem kurz beschrieben.
Mit „Load Network“ kann man ein beliebiges Configuration-File ins Programm laden, wo dann das Netzwerk im Bereich zur Darstellung des Netzwerks visualisiert wird.
Bei „Save Network“ wird dann das geladene Netzwerk als eine XML-Datei raus geschrieben. Natürlich werden alle Konfigurationen des Netzwerks wie zum Beispiel die Positionen der einzelnen Komponenten in dieser Datei gespeichert. Die erzeugte XML-Datei kann man auf Wunsch jederzeit wieder ins Programm laden.
Der Menüpunkt „Exit“ beendet das gesamte Programm. Dies ist der einzige Menüpunkt, der zu jeder Zeit betätigt werden kann.

Unter dem zweiten Oberpunkt verbergen sich alle Steuerungselemente, mit denen man den Spanning Tree Algorithmus bedienen und verwalten kann. Hier befinden sich die Menüpunkte „Start Simulation“, „Pause Simulation“, „Stop Simulation“, „SAT-Mac Tables“ und „Configurations“. All diese sind jedoch erst nach dem erfolgreichen Laden eines Configuration-Files zugänglich.
Mit den ersten drei kann man den Spanning Tree Algorithmus starten, pausieren und beenden. Da sich jeder etwas unter ihnen vorstellen kann, wird hier nicht näher darauf eingegangen.
Mit dem Menüpunkt „SAT-Mac Tables“ kann man sich die SAT-Mac Table zu jedem Switch anzeigen lassen. Zu Beginn sind sie natürlich noch alle leer, da noch keine Nutzdaten zwischen den Hosts verschickt wurden. Wenn man lange genug wartet, werden sich diese jedoch mit Inhalt füllen. Dann weiß jeder Switch über welchen Port er jeden beliebigen Host des Netzwerks erreichen kann. Diese Funktion steht dem Nutzer während dem laufenden Algorithmus nicht zu Verfügung. Er muss diesen dazu vorher zumindest auf Pause schalten. Wer mehr über dieses Fenster erfahren möchte, muss in Kapitel 5.2 auf Seite 32 weiterlesen.
Der letzte Menüpunkt „Configurations“ bietet die Möglichkeit die Switches und ihre Ports nachträglich zu konfigurieren. Damit ist gemeint, dass man viele der Attribute, die man für diese beiden Komponenten-Arten in dem Configuration-File angegeben hat, im laufenden Programm verändern kann. Natürlich kann man den grundlegenden Aufbau des Netzwerks hier nicht mehr verändern, aber man kann zum Beispiel die Prioritäten oder die Pfadkosten nach Belieben anpassen. Man hat sogar die Möglichkeit vor dem Starten des Algorithmus noch beliebige Ports auf disabled zu setzen. Dies kann man jedoch ausschließlich vor dem

Starten des Algorithmus tun. Eine weitere kleine Einschränkung gibt es jedoch noch. Man kann auch die anderen Einstellungen unter Configurations nicht während dem laufenden Algorithmus machen. Man muss diesen zumindest auf Pause schalten. Auch zu diesem Fenster findet man später eine ausführlichere Beschreibung und zwar in Kapitel 5.1 auf Seite 28.

Der letzte Oberpunkt „Help“ beinhaltet den Menüpunkt „Manual“. Dieser ermöglicht es dem Nutzer ein Handbuch zu diesem Programm aufzurufen. Dieses Handbuch liegt als PDF-Dokument vor und beschreibt die Bedienung aller Funktionalitäten und Schaltflächen des Programms.

4.2 Kopfbereich

Der Kopfbereich des Hauptfensters ist, der obere von den drei großen Bereichen, aus denen das Hauptfenster besteht. In Abbildung 07 kann man ihn leicht finden, da er in diesem Screenshot rot umrandet wurde.

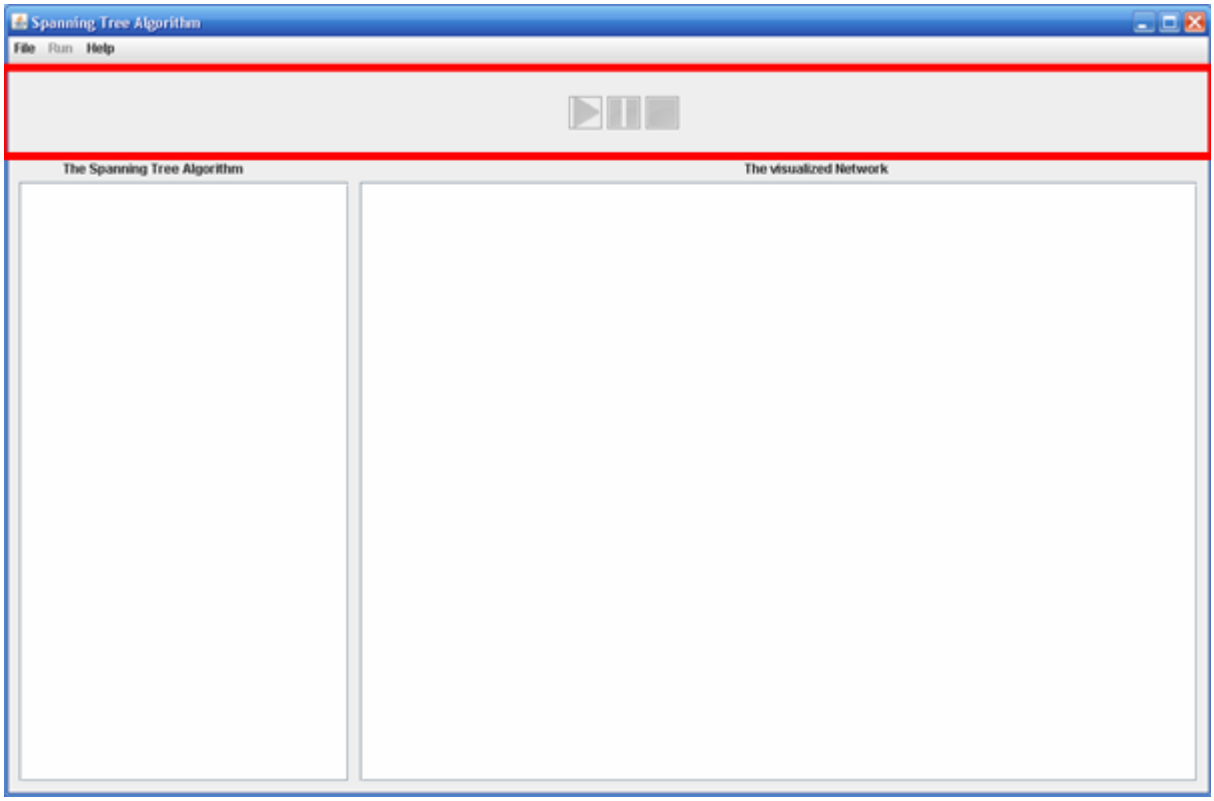

Abbildung 07: Kopfbereich im Hauptfenster

Abbildung 07 zeigt das Hauptfenster des Programms direkt nach dem Starten des Programms. Deshalb befinden sich im Kopfbereich momentan nur die drei Buttons zum Starten, Pausieren und Stoppen des Spanning Tree Algorithmus. Aber wie man sehen kann, sind diese Buttons momentan noch nicht aktiv, da ja noch kein Configuration-File geladen wurde, auf der man den Algorithmus laufen lassen könnte.

Sobald man dies aber getan hat, verändert sich der Kopfbereich. Es kommen neue Funktionen hinzu, was man in Abbildung 08 erkennen kann.

Abbildung 08: Kopfbereich nach dem Laden eines Configuration-Files

Auch in Abbildung 08 sind wieder die drei Buttons vorhanden, jedoch ist nun der Start-Button aktiv, da ein Configuration-File erfolgreich ins Programm geladen wurde. Links davon befindet sich jetzt eine Schaltfläche, die einem die Möglichkeit bietet, das Altern der Einträge in den SAT-Mac Tables an- bzw. abzuschalten. Hierauf wird auch erst in Kapitel 5.2 auf Seite 32 eingegangen. Am rechten Rand des Kopfbereichs ist auch noch die Legende aufgetaucht, in der die verschiedenen Schriftfarben mit ihren Bedeutungen für die Visualisierung des Netzwerks aufgelistet sind. An der Legende wird sich von nun an nichts mehr verändern.

Sollte man den Spanning Tree Algorithmus nun starten, verändert sich der Kopfbereich ein weiteres mal. Diese Veränderung ist auf Abbildung 09 zu sehen.

Abbildung 09: Kopfbereich nach dem Starten des Algorithmus

Die linke Schaltfläche zum An- bzw. Abschalten der Alterung wurde nun, wie in Abbildung 09 zu sehen, deaktiviert. Auch wurde der Start-Button deaktiviert, da man den Algorithmus nicht zweimal starten könne soll. Dafür wurden aber der Pause- und der Stopp-Button aktiviert.

Die letzte Veränderung am Kopfbereich findet beim Schalten auf Pause statt. Auch hierzu gibt es wieder eine kleine Veranschaulichung und zwar Abbildung 10.

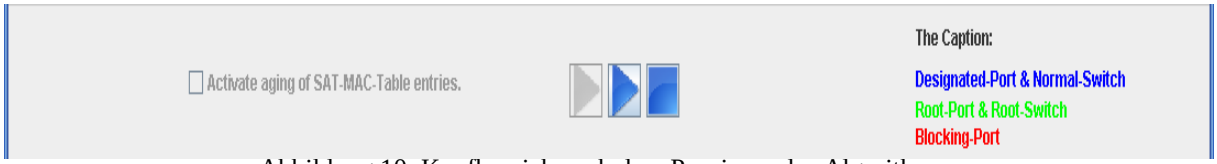

Abbildung 10: Kopfbereich nach dem Pausieren des Algorithmus

In Abbildung 10 kann man sehen, dass sich nur der Pause-Button verändert hat. Da er ja, um den Algorithmus auf Pause zu schalten, schon einmal angeklickt wurde, zeigt er jetzt dasselbe Symbol wie der Start-Button, damit man erkennen kann, dass der Algorithmus auf Pause geschaltet wurde. Beim erneuten Anklicken wechselt das Symbol wieder zum ursprünglichen. Wenn man den Algorithmus irgendwann beendet, sieht der Kopfbereich wieder genauso aus, wie er in Abbildung 08 zu sehen ist.

4.3 Bereich zur Darstellung des Netzwerks

Der Bereich zu Darstellung des Netzwerks ist der rechte der vier Bereiche. Er ist im Programm durch dir Überschrift „The visualized Network“ eindeutig zu identifizieren. In Abbildung 11 kann man ihn zusätzlich zur Überschrift auch noch an der roten Umrandung erkennen.

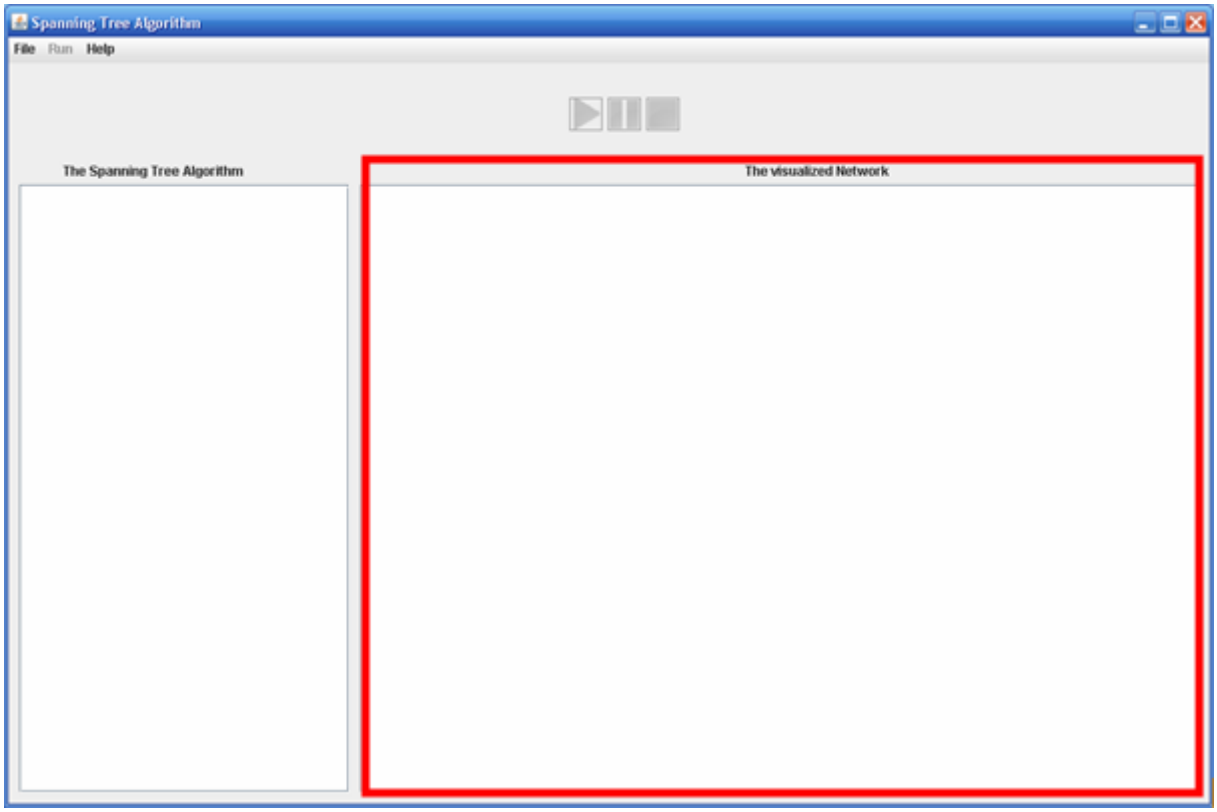

Abbildung 11: Bereich zu Darstellung des Netzwerks

Wie in Abbildung 11 zu sehen, ist der Bereich zur Darstellung des Netzwerks direkt nach dem Starten des Programms noch leer. Um diesen zu füllen, muss man ein Configuration-File laden. Sobald man dies getan hat, wird das Netzwerk sofort automatisch in diesem Bereich visualisiert. Da es zu Realisierung eines Netzwerks ein eigenes Kapitel gibt, nämlich Kapitel 6 auf Seite 36, wird hier auf diesen Bereich nicht näher eingegangen. Aber wegen der Vollständigkeit wurde er hier trotzdem einmal kurz gezeigt.

4.4 Bereich zur Darstellung des Spanning Tree Algorithmus

Da der Nutzer dieses Programms auch etwas von der Arbeit des Algorithmus sehen möchte, wurde neber der farblichen Kennzeichnung der Port- bzw. Switch-Arten auch ein extra Bereich im Hauptfenster angelegt, wo die beiden wichtigsten Informationen jedes Switches, nämlich die Root ID und die Root-Pfadkosten angezeigt werden. Dies sind für den Algorithmus die beiden entscheidenden Informationen. Alle anderen verändern sich im Laufe des Algorithmus nicht und wurden deshalb auch nicht dargestellt.

Diesen Bereich kann man einmal durch die Überschrift „The Spanning Tree Algorithm“ und zweitens durch die rote Umrandung in der Abbildung 12 identifizieren.

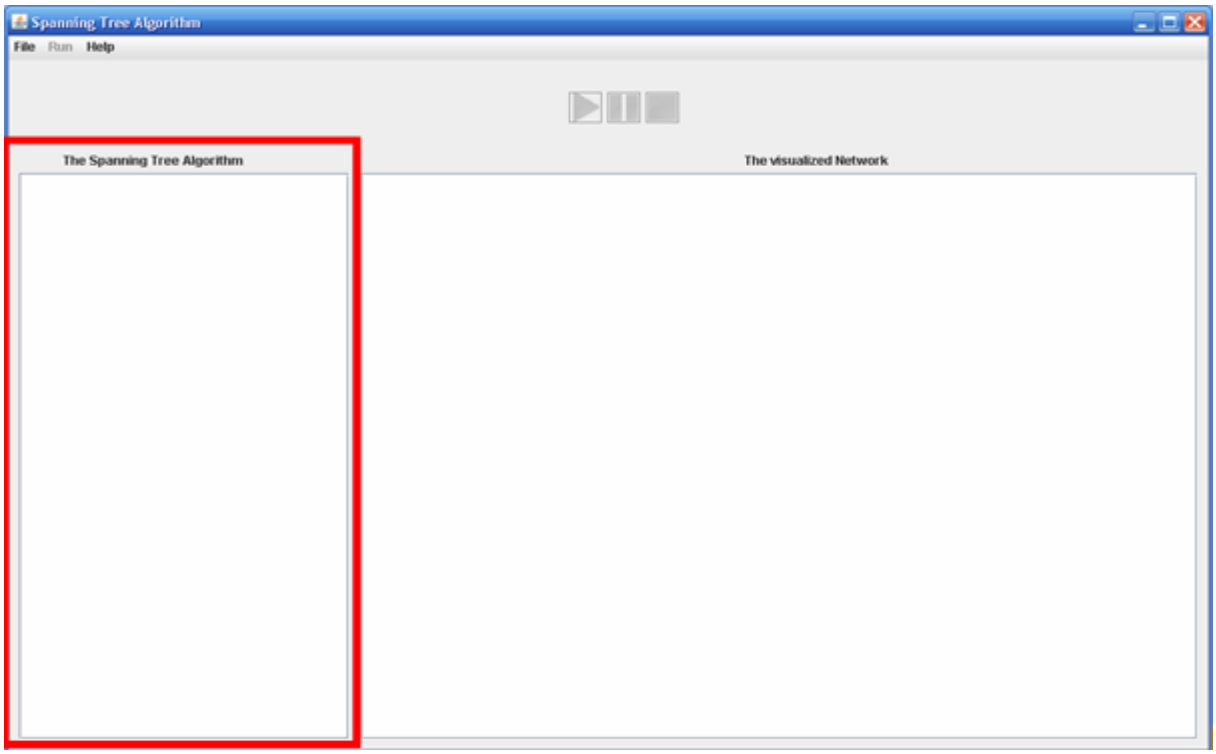

Abbildung 12: Bereich zur Darstellung des Spanning Tree Algorithmus

Wie auch der Bereich zur Darstellung des Netzwerks ist dieser zu Beginn leer. Da die beiden Fakten, die dort angezeigt werden sollen, erst mit dem Start des Algorithmus mit Werten gefüllt werden, wird auch erst dann etwas in diesem Bereich zu sehen sein. Deshalb folgt nun ein weiterer Screenshot und zwar Abbildung 13, auf dem man sich einmal anschauen kann, wie dieser Bereich aussieht, wenn dort das Wissen der Switches angezeigt wird.

Abbildung 13: Bereich zur Darstellung des Spanning Tree Algorithmus

In Abbildung 13 kann man einen Teil des Wissens jedes Switches sehen und zwar die Root ID und die Root-Pfadkosten. Was diese beiden Werte bedeuten, wurde in Kapitel 2 auf Seite 3 schon beschrieben. Da es sich hier wieder um das Configuration-File 01 handelt, besteht das Netzwerk wieder nur aus vier Switches. Nachdem der Spanning Tree Algorithmus fertig ist, weiß jeder der vier Switches, dass der Switch 001 der Root-Switch ist, was man auch hier ablesen kann. Auch die Root-Pfadkosten wurden von allen Switchen richtig bestimmt und hier ausgegeben. Was einem in dieser Abbildung noch auffällt, sind die vier Buttons. Diese sind während dem laufenden Algorithmus deaktiviert. Erst im Pause-Modus oder nach dem Beenden des Algorithmus kann man sie anklicken. Sollte man dies tun, öffnet sich ein neues Fenster, in dem man sich alle BPDUs anschauen kann, die dieser Switch empfangen hat. Dieses Fenster wird aber erst in Kapitel 5 auf Seite 28 näher beschrieben.

Zum Abschluss dieses Kapitels wird in Abbildung 14 einmal das ganze Hauptfenster nach dem Beenden des Spanning Tree Algorithmus gezeigt. Nun kann sich jeder ein Bild von diesem Hauptfenster machen und es dürfte nun hoffentlich klar sein, wie es aufgebaut ist und welche Möglichkeiten es bietet.

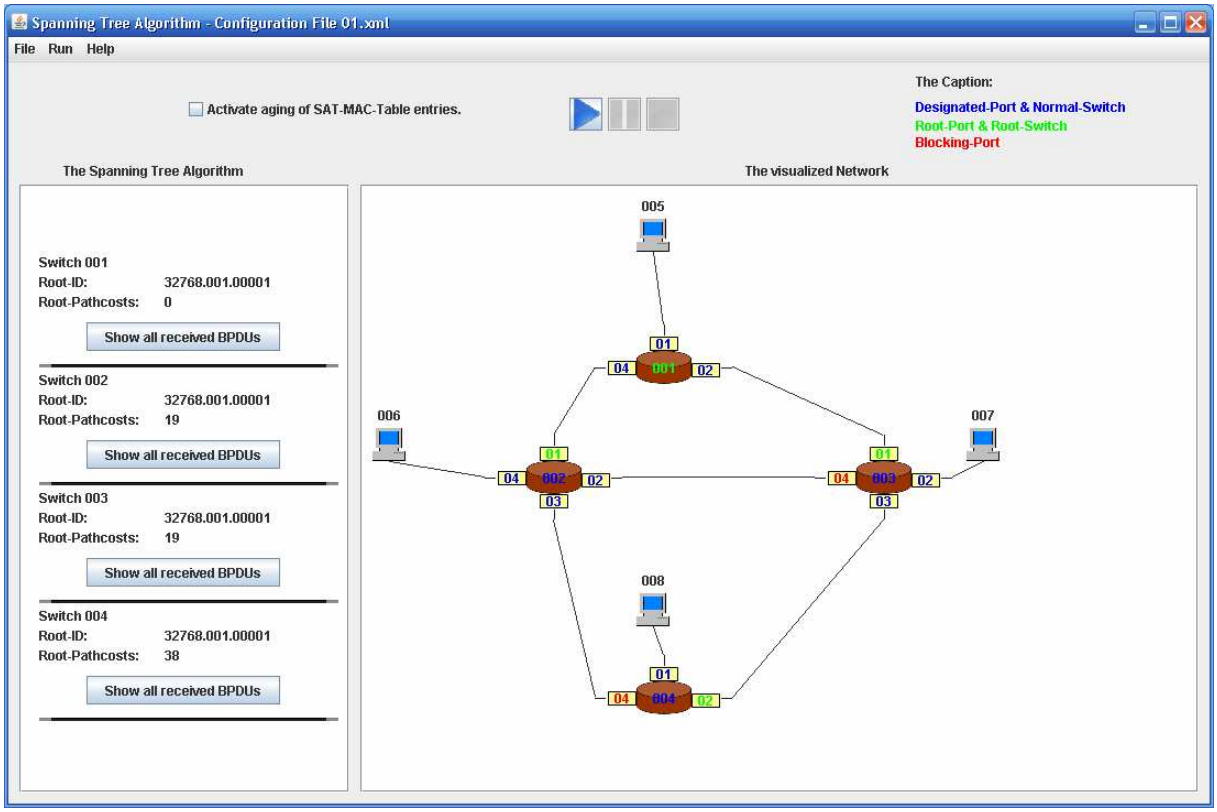

Abbildung 14: Hauptfenster nach Beendigung des Spanning Tree Algorithmus

5 Die Nebenfenster

Dieses Programm enthält außer dem Hauptfenster noch drei weitere wichtige Fenster, die in Kapitel 5 behandelt und zusammen als Nebenfenster bezeichnet werden. Diese sind „Configurations“, „SAT-Mac Tables“ und „All received BPDUs“. Im ersten kann der Nutzer alle Attribute der einzelnen Komponenten wie die Switches mit ihren Ports und die Hosts einsehen und einige davon sogar nach Belieben verändern. Das zweite Fenster zeigt von allen Switchen deren jeweilige SAT-Mac Table und das dritte alle von einem Switch empfangenen BPDUs an.

5.1 Configurations

Zuerst wird erst einmal das Nebenfenster „Configurations“ näher beschrieben. Dieses zeigt zu allen Komponenten des Netzwerks ihre jeweiligen Attribute an, von denen man einige nachträglich verändern kann. Wie vorher schon einmal erwähnt wurde, kann man diese Funktionalität des Programms erst nutzen, wenn man bereits ein Configuration-File geladen hat, da es schließlich vorher noch keine Komponenten gibt, von denen man sich irgendwelche Attribute anzeigen lassen könnte.
Sollte man nun den Menüpunkt „Configurations“ anklicken, öffnet sich ein kleines Fenster, in dem man sich entscheiden muss, ob man sich die Attribute der Switches oder der Hosts anschauen möchte. In Abbildung 15 kann man sich dieses einmal anschauen.

Abbildung 15: Entscheidung zwischen Switch und Host

Wenn man sich für die Switches entschieden hat, öffnet sich sofort ein zweites Fenster, in dem alle Switches mit ihren dazugehörigen Ports aufgelistet sind. Diese Auflistung kann man in Abbildung 16 einsehen.

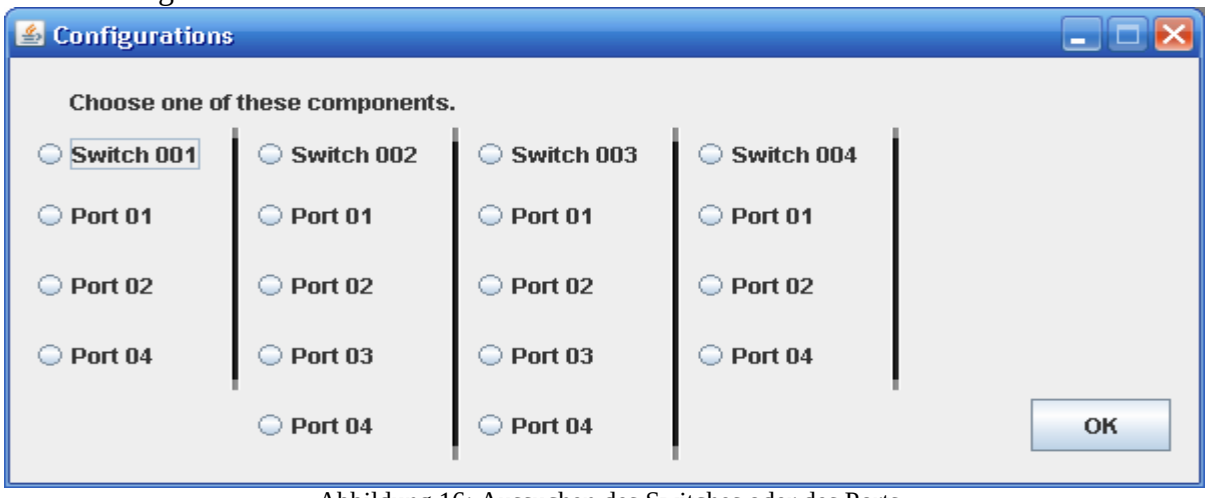

Abbildung 16: Aussuchen des Switches oder des Ports

Die Abbildung 16 ist eigentlich nicht schwer zu verstehen, aber um Missverständnisse auszuschließen sollte hier kurz erwähnt werden, damit die Ports unterhalb eines Switches auch zu diesem gehören. Nun muss man sich entscheiden, welche der angezeigten Komponenten man sich näher betrachten möchte.
Sollte die Wahl wiederum auf einen der Switches gefallen sein, öffnet sich noch ein drittes Fenster, wo man nun endlich die Attribute des ausgewählten Switches betrachten kann. Als Beispiel wurde hier Switch 001 gewählt, dessen Attribute man in Abbildung 17 sehen kann.

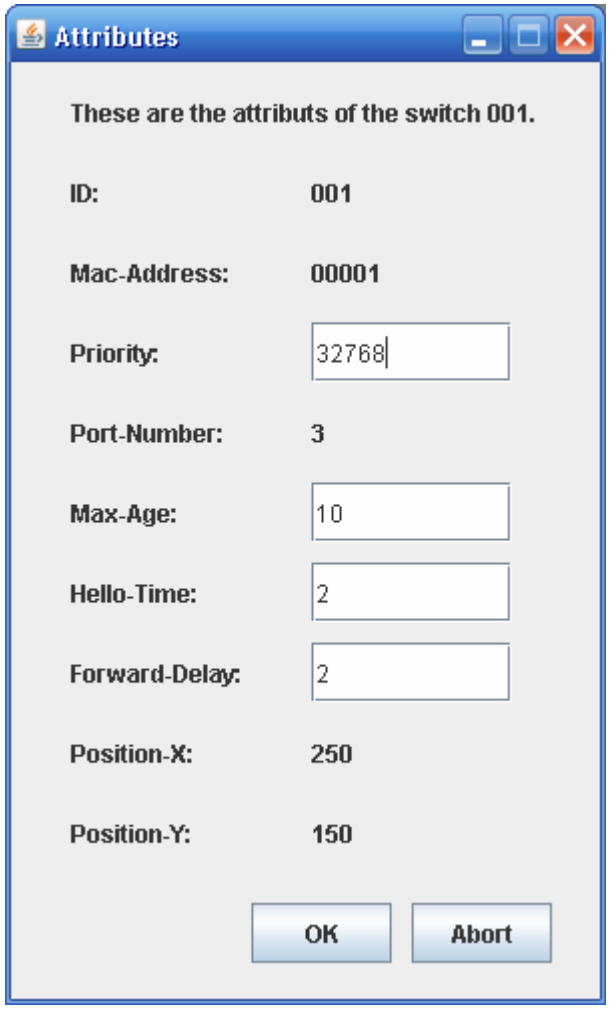

Abbildung 17: Attribute von Switch 001

Wie man in Abbildung 17 erkennen kann, werden die verschiedenen Attribute auf zwei unterschiedlichen Arten dargestellt. Es gibt die unveränderlichen Attribute wie ID, Mac-Adresse und Port-Number. Diese drei lassen sich ausschließlich innerhalb des Configuration-Files angeben und verändern, da sie zum Aufbau und zur Visualisierung des Netzwerks notwendig sind. Die beiden letzten Attribute Position-X und Position-Y lassen sich zwar in diesem Fenster auch nicht verändern, aber der Nutzer hat die Möglichkeit, die Darstellungen der Switches und der Hosts innerhalb des Bereichs zur Darstellung des Netzwerks mittels Drag&Drop zu verschieben. Auf diese Möglichkeit wird in Kapitel 6 auf Seite 36 näher eingegangen. Dann folgen jetzt noch die Attribute, die sich innerhalb dieses Fensters verändern lassen, da sie ausschließlich für den Spanning Tree Algorithmus notwendig sind. Aus diesem Grund kann man die Configurations auch nur öffnen, wenn der Algorithmus nicht läuft, also wenn er beendet oder auf Pause geschaltet wurde. Diese veränderlichen Attribute sind Priority, Max-Age, Hello-Time und Forward-Delay. Wofür diese stehen, wurde bereits in Kapitel 2 auf Seite 3 erklärt.

Sollte man sich in Abbildung 16 jedoch für einen Port entschieden haben, öffnet sich natürlich ein anderes Fenster, das man in Abbildung 18 sehen kann.

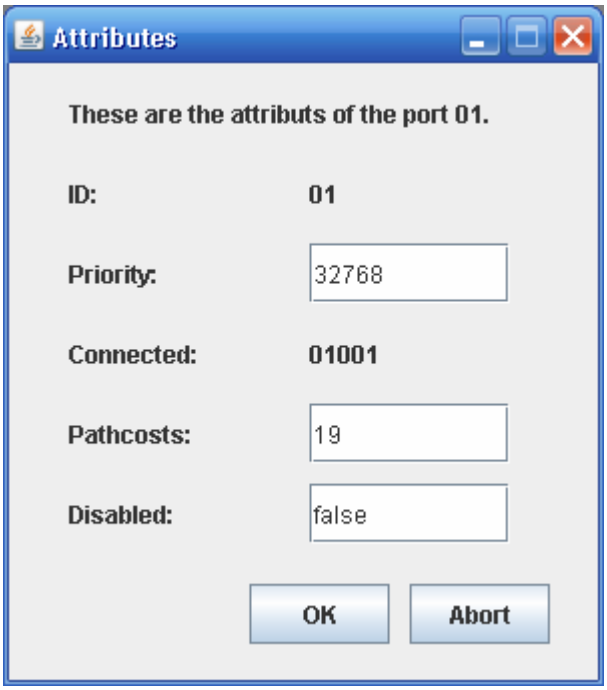

Abbildung 18: Attribute von Port 01 von Switch 001

Wie man in Abbildung 18 sehen kann, haben die Ports viel weniger Attribute als die Switches. Auch hier gibt es wieder unveränderliche und veränderliche Attribute. Zu der ersten Kategorie gehören die ID und Connected. Im Feld Connected steht die Mac-Adresse des Hosts bzw. die ID des Ports, mit dem dieser Port mittels Wire verbunden ist. In diesem Beispiel ist der Port 01 des Switches 001 mit dem Host 005, der die Mac-Adresse 01001 hat, verbunden. Zur zweiten Kategorie gehören die Priority, die Pathcosts und Disabled. Solange disabled auf false steht, ist der Port aktiv. Sollte dieses Attribut jedoch auf true gesetzt werden, wird dieser Port sofort deaktiviert. Was dies für den Algorithmus bedeutet, wurde in Kapitel 2 auf Seite 3 schon erklärt. Jedoch gibt es hier eine kleine Einschränkung. Da die Rekonfiguration des Spannbaums nicht Bestandteil dieser Bachelorarbeit ist, kann man das Attribut disabled nur vor dem Starten bzw. nach dem Beenden des Algorithmus verändern, jedoch nicht falls der Algorithmus auf Pause geschaltet ist. Um dies sicherzustellen, wird das Eingabefeld bei disabled im Pausemodus deaktiviert.
Da eine Verbindungsleitung einen Anfang und ein Ende hat, sind all jene, die zwei Ports miteinander verbinden, zweimal innerhalb eines Configuration-Files definiert und zwar einmal beim Startport und einmal beim Endport. Um zu verhindern, dass man im Programm zweimal dieselbe Wire hat, wird nur die Definition vom Startport genommen. Diese Entscheidung führt jedoch hier zu einer kleinen Unschönheit. Das Attribut Connected ist nicht bei allen Ports vorhanden. Es liegt aus dem eben genannten Grund nur bei den Startports und nicht bei den Endports vor. Um dieses Feld nicht einfach leer zu lassen, werden anstelle des eigentlichen Werts drei x angezeigt.

Dies kann man in Abbildung 19 sehen.

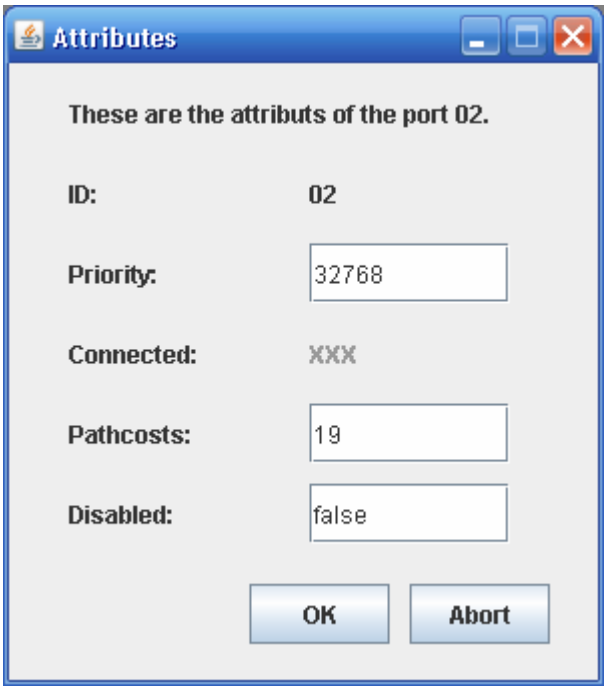

Abbildung 19: Attribut Connected mit xxx

Um doch noch rauszubekommen, mit welchem anderen Port der gewählte verbunden ist, schaut man am Besten in der Visualisierung des Netzwerks nach.

Als letzten Punkt innerhalb dieses Unterkapitels bleiben nun nur noch die Hosts übrig. Sollte man sich ganz zu Beginn in Abbildung 15 anstatt für die Switches für die Host entschieden haben, erscheint natürlich auch ein neues Fenster. In diesem sind alle Hosts des Netzwerks aufgelistet, von denen man sich einen aussuchen kann. Dies ist in Abbildung 20 zu erkennen.

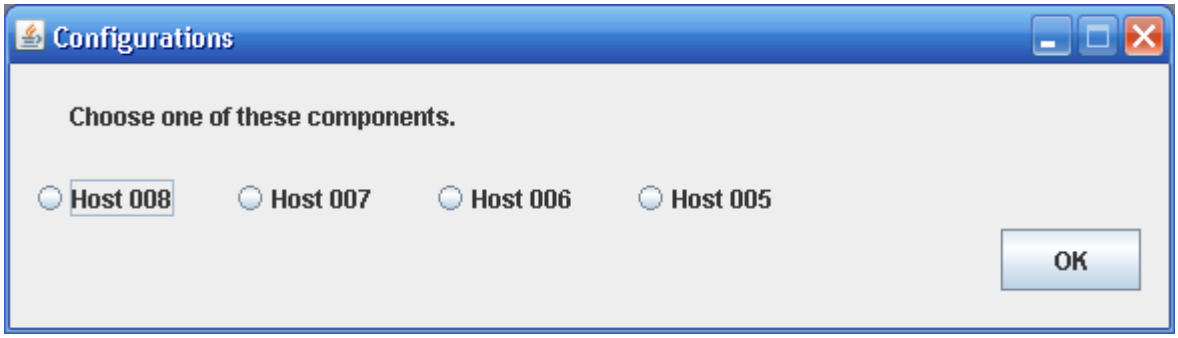

Abbildung 20: Hostauswahl

Wie man in Abbildung 20 sehen kann, gibt es in diesem Netzwerk vier Hosts. Wenn man sich für einen dieser vier entschieden hat, öffnet sich erneut ein Fenster. In diesem sind dann alle Attribute des jeweiligen Hosts aufgelistet. Anders als bei den Switchen gibt es hier nur unveränderliche Attribute.

Dies kann man in Abbildung 21 betrachten.

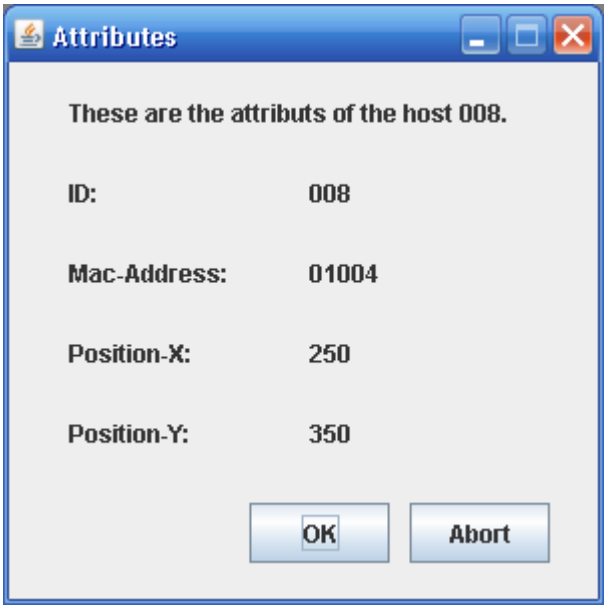

Abbildung 21: Attribute des Hosts 008

Auch die Hosts haben im Vergleich zu den Switchen recht wenige Attribute und zwar eine ID, eine Mac-Adresse und natürlich die beiden Positionsangaben Position-X und Position-Y. Diese beiden kann man in diesem Fenster nicht verändern, aber man kann die Visualisierung der Host im Bereich zur Darstellung des Netzwerks mittels Drag&Drop verschieben. Auf diese Weise kann man dann die Positionsangaben ändern.

5.2 SAT-Mac Tables

In diesem Unterkapitel wird erklärt, was eine SAT-Mac Table ist und wie sie im Programm aufgebaut ist.
Jeder Switch besitzt solch eine Tabelle, die zu Beginn natürlich noch leer ist. Um eins direkt klarzustellen: Sie ist kein Teil des Spanning Tree Algorithmus und wird von diesem auch in keinster Weise verwendet. Die eigentliche Aufgabe eines Netzwerks ist es, Nutzdaten von einem Host zu einem anderen zu verschicken. Wie aber soll dies geschehen, wenn kein Switch weiß, wo sich der Ziel-Host befindet? Er macht einfach einen Broadcast über all seine Ports. Dies ist eine sehr ungünstige Variante, Nutzdaten zum Ziel zu verschicken, da dadurch große Teile des Netzwerks vollkommen unnötig beansprucht und ausgebremst werden. Aus diesem Grund merkt sich ein Switch, sobald er Nutzdaten empfängt, vom welchem Host diese kommen und über welchen Port er sie empfangen hat. Genau diese Zuordnung merkt er sich in seiner SAT-Mac Table. Trotzdem weiß ein Switch zu Beginn nicht, wo sich der Ziel-Host befindet. Deshalb kommt er am Anfang um Broadcast nicht herum. Aber mit der Zeit füllt sich seine Tabelle und er kann empfangene Nutzdaten immer öfter gezielt verschicken, da er weiß, über welchen Port er den Ziel-Host erreichen kann. Wie in Kapitel 2 auf Seite 3 beschrieben, werden Nutzdaten nur von Ports, die sich im Forwarding-Status befinden, empfangen und weitergeleitet. In echten Netzwerken können ja Switches, Verbindungsleitungen oder Hosts ausfallen. Deshalb wurde eine Alterung der Einträge dieser Tabelle realisiert. Das bedeutet, dass jeder Eintrag in dieser Tabelle mit einem Alter versehen ist. Sollte ein Eintrag irgendwann zu alt werden, wird er aus der Tabelle wieder gelöscht. Wie in Kapitel 4 auf Seite 22 erwähnt, bietet dieses Programm die Möglichkeit diese Alterung an- bzw. abzuschalten. Denn wenn sich jemand einmal die vollständige SAT-Mac Table

anschauen möchte, dürfte dies bei alternden Einträgen schwierig werden, da immer irgendwelche Einträge gerade nicht in dieser enthalten sind.
Nun dürfte klar sein, was eine SAT-Mac Table ist und wozu sie benötigt wird. Diese Tabellen werden erst beim Starten des Algorithmus erstellt, die aber zu Beginn noch leer sind. Anschauen kann man sich diese, wenn man den Algorithmus pausiert oder beendet. Möchte man die Tabellen komplett ausgefüllt sehen, muss man den Algorithmus aber auch einige Zeit laufen lassen, damit alle Nutzdaten ihr Ziel erreichen können. Beim Netzwerk aus Configuration-File 01 dürfte dies so ungefähr zwei Minuten betragen.
Nachdem man den Menüpunkt „SAT-Mac Tables“ angewählt hat, öffnet sich ein neues Fenster, wo alle Switches des Netzwerks aufgelistet sind. Nun muss man sich für einen davon entscheiden, um dessen SAT-Mac Table anschauen zu können. Dies kann man in Abbildung 22 erkennen.

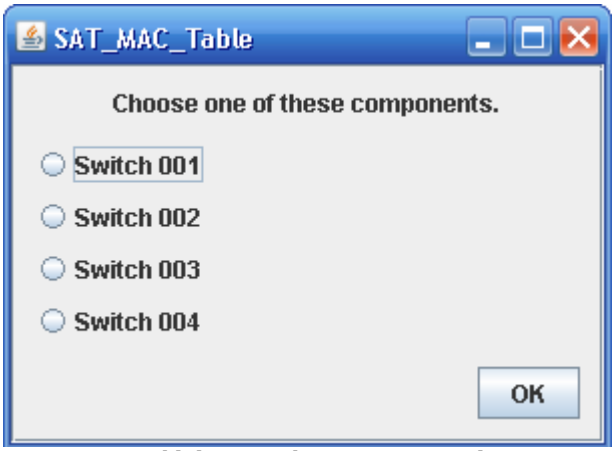

Abbildung 22: Auswahl des Switches zur Anzeige der SAT-Mac Table

Egal für welchen Switch man sich nun entscheidet, es öffnet sich jedes Mal ein neues Fenster mir der entsprechenden SAT-Mac Table. In Abbildung 23 und 24 kann man erkennen, dass sich die Anordnung der Host-IDs von Switch zu Switch stark unterscheiden kann.

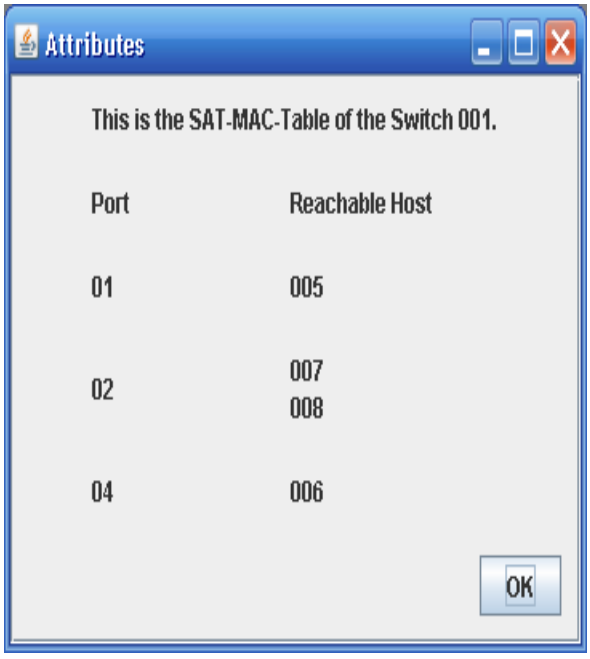

Abbildung 23: SAT-Mac Table von Switch 001

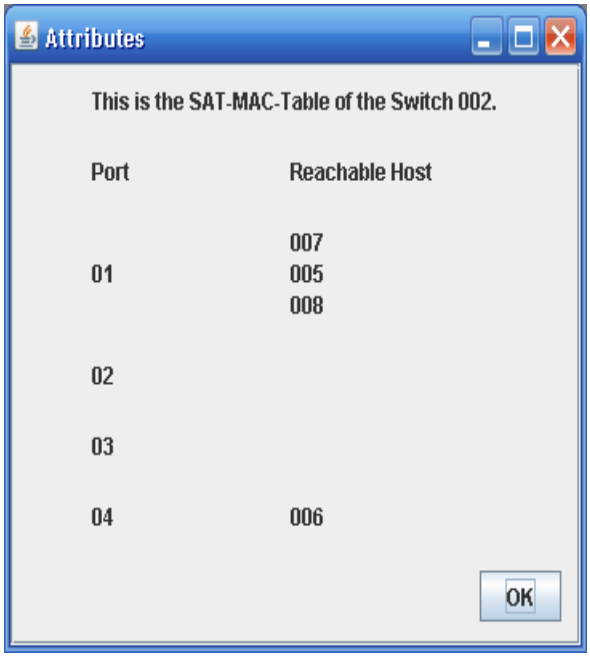

Abbildung 24: SAT-Mac Table von Switch 002

Wie man anhand dieser beiden Abbildungen sehen kann, besteht jede SAT-Mac Table aus zwei Spalten. In der linken befinden sich alle Ports des entsprechenden Switches und in der rechten sind zu jedem Port die von dort erreichbaren Hosts mit ihren IDs eingetragen. Es kann also, wie man in Abbildung 24 sieht, auch Ports geben, über die man überhaupt keine Hosts erreichen kann. Dies sind in diesem Beispiel Port 02 und 03 des Switches 002. Über Port 04 erreicht man den Host, der direkt an diesen Switch angeschlossen ist und über Port 01 alle anderen Hosts des Netzwerks.

5.3 All received BPDUs

Mittlerweile wurden also die beiden Nebenfenster „Configurations“ und „SAT-Mac Tables“ beschrieben. Jetzt fehlt nur noch das Nebenfenster „All received BPDUs“, das über das Hauptfenster direkt aufgerufen wird und alle empfangenen BPDUs eines Switches anzeigt.
Das Wissen über den Aufbau einer BPDU wird hier vorausgesetzt, da er bereits in Kapitel 2 auf Seite 3 ausführlichst erklärt wurde. Wie aus dem eben genannten Kapitel bekannt sein dürfte, besteht eine solche Nachricht aus vielen Fakten, die aus Platzgründen nicht alle auf einen Screenshot passen. Deshalb werden nur die entscheidenden Teile mit Screenshots veranschaulicht. Sobald man den Spanning Tree Algorithmus startet, wird der Bereich zur Darstellung des Spanning Tree Algorithmus mit Informationen gefüllt. Unter anderem erscheint dort zu jedem Switch ein Button, mit dessen Hilfe man das in diesem Unterkapitel beschriebene Nebenfenster aufrufen kann. Verständlicherweise sollte man den Algorithmus erst einmal eine Zeit laufen lassen, damit auch genügend BPDUs erzeugt und empfangen wurden. Denn vorher lohnt es sich nicht wirklich, dieses Fenster aufzurufen.
In Kapitel 2 auf Seite 3 wurde ja der Spanning Tree Algorithmus anhand des Netzwerks aus Configuration-File 01 ausführlich beschrieben. Deshalb dürfte bekannt sein, dass Switch 001 der Root-Switch ist und nur er BPDUs erzeugen darf. Wenn man sich den Aufbau des Netzwerks mit dem Wissen über den Algorithmus im Hinterkopf etwas näher betrachtet, wird einigen auffallen, dass Switch 001 nur BPDUs empfangen darf, solange Switch 002 und Switch 003 davon ausgehen, dass sie der Root-Switch sind. Sobald sie wissen, dass Switch 001 diese Stellung inne hat, dürfen sie keine BPDUs mehr an Switch 01 schicken. Dies kann man auf der Abbildung 25 sehen.

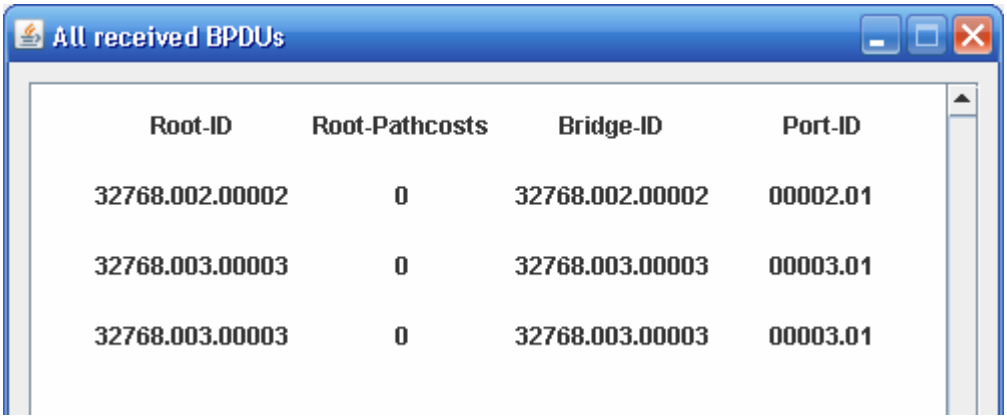

Abbildung 25: Empfangene BPDUs von Switch 001

Der Switch 001 hat nur drei BPDUs und zwar von Switch 002 und Switch 003 empfangen. Egal wie lange man den Algorithmus jetzt noch laufen lassen würde, es würde bei diesen drei BPDUs bleiben. Es müssen nicht bei jedem Durchlauf immer exakt drei BPDUs an Switch 001 verschickt werden. Es könnten manchmal auch ein paar mehr als nur drei sein, aber es werden immer ziemlich wenige bleiben. In Abbildung 25 ist nur der relevante Teil der BPDUs abgebildet. Würde man nach links bzw. nach rechts scrollen, könnte man die

komplette BPDU mit all ihren Fakten sehen. Wenn man sich zum Vergleich einmal die empfangenen BPDUs von Switch 002 anschaut, merkt man sofort, dass dieser bedeutend mehr Nachrichten empfangen hat und auch immer wieder neue empfangen wird, da der Root-Switch alle zwei Sekunden neue BPDUs erzeugt und verschickt. Dies kann man in Abbildung 26 erkennen.

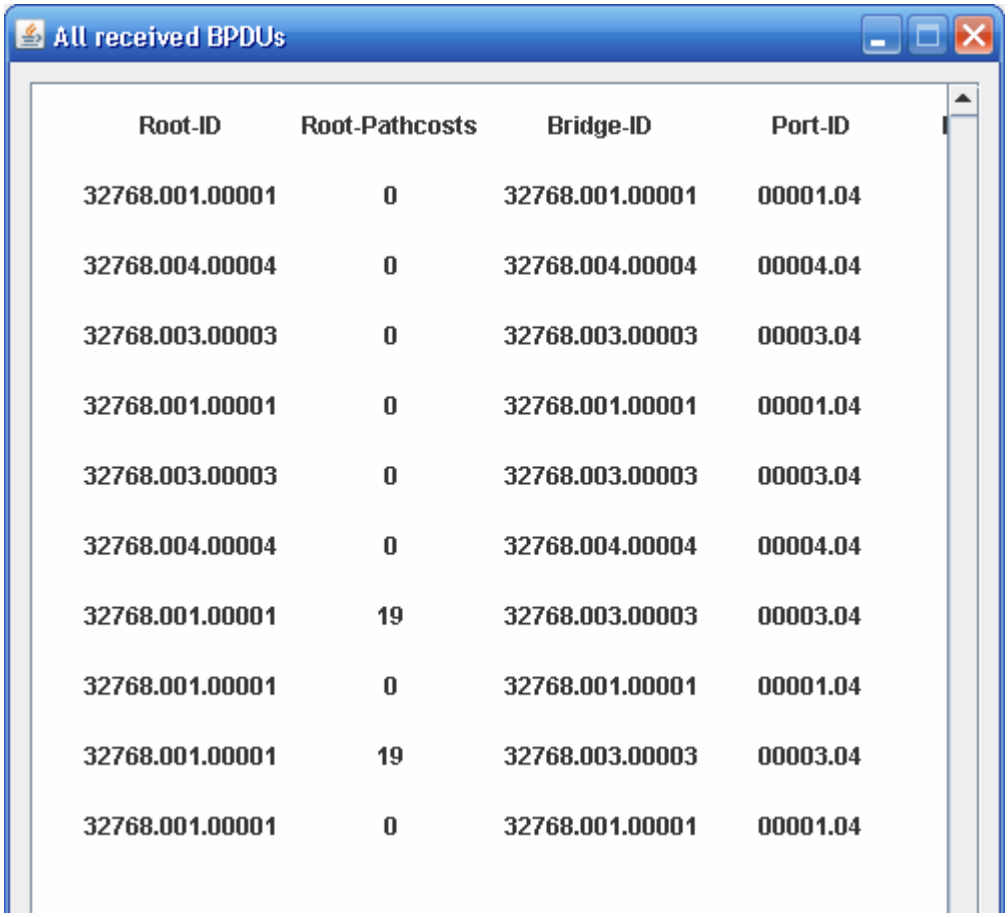

Abbildung 26: Empfangene BPDUs von Switch 002

In Abbildung 26 kann man gut erkennen, wie viele BPDUs Switch 002 empfangen hat. Würde man den Algorithmus länger laufen lassen, erhöht sich diese Zahl stetig und es stellt sich heraus, dass Switch 002 nur noch von Switch 001 BPDUs empfangen wird, da alle anderen Wege blockiert werden. Was als letztes noch erwähnt werden sollte, ist, dass alle Blocking-Ports zwar keine BPDUs aussenden aber empfangen dürfen. Deshalb empfängt beispielsweise Switch 003 über Port 04 BPDUs von Switch 002 aber nicht umgekehrt.

6 Realisierung eines Netzwerks

In Kapitel 6 geht es ausschließlich um die Realisierung eines Netzwerks im Rahmen dieser Bachelorarbeit. Zuerst werden die Bestandteile erklärt, die in Netzwerken verwendet werden dürfen. Natürlich werden nur Netzwerke betrachtet, für die das Programm auch ausgelegt ist. Im Anschluss wird erklärt, wie man sich solch ein Netzwerk mit Hilfe eines Configuration-Files erstellen kann und auf was man dabei alles zu achten hat. Zum Schluss wird dann noch gezeigt, wie solch ein Netzwerk im Programm visualisiert wird.

6.1 Bestandteile eines Netzwerks

In diesem Unterkapitel werden erst einmal alle erlaubten Bestandteile vorgestellt, die in hier verwendeten Netzwerken benutzt werden. Die beiden wichtigsten sind, wie schon vorher mehrfach erwähnt, die Switches und die Hosts. Die Anzahl dieser beiden Komponenten-Arten kann natürlich beliebig variieren. Jedoch muss jeder Host immer an genau einen Switch angeschlossen werden. Bei den Switches wird von Layer 2 Switches ausgegangen. Um das Ganze noch etwas realistischer zu gestalten, hat jeder Switch auch noch Ports bekommen. Diese stellen keine eigenen Komponenten dar, sondern sie sind Bestandteil des jeweiligen Switches. Auch ihre Anzahl kann beliebig variieren. Für die Verbindung zwischen zwei Komponenten wird ein Kabel, im weiteren Wire genannt, verwendet. An jedem Port muss immer genau eine Wire angeschlossen sein. Ebenfalls darf es keine Komponenten geben, die nicht angeschlossen sind. Das bedeutet, dass jede Komponente durch eine Wire mit einer anderen verbunden sein muss.
Jeder Switch kann vom Nutzer im laufenden Programm verwaltet und angepasst werden. Wie vorher schon erwähnt, läuft auf dieser Komponenten-Art der Spanning Tree Algorithmus, weshalb sie BPDUs erzeugen, verschicken und empfangen können. Sie legen sich zudem noch jeweils eine SAT-Mac Table an, die ihnen beim gezielten Verschicken von Nutzdaten hilft.
Die einzige Aufgabe der Hosts ist es, Nutzdaten zu anderen Hosts zu schicken. Sie sind nicht Bestandteil des Spanning Tree Algorithmus.

6.2 Aufbau eines Configuration-Files

Das Unterkapitel 6.2 befasst sich mit dem Aufbau eines Configuration-Files. Hier wird Schritt für Schritt erklärt, wie man sich solch eine Datei selber erstellen kann und worauf man zu achten hat. Am Ende dieses Kapitels hat man das Configuration-File 01 nachgebaut.
Man verwendet zur Erstellung solch eines Configuration-Files am Besten den Editor, der von Windows direkt mitgeliefert wird. Die Sprache, in der die Datei erstellt wird, ist XML. Aber auch jemand, der dieser Sprache nicht mächtig ist, wird nach dieser sehr ausführlichen Erklärung in der Lage sein sich ein Netzwerk mit Hilfe eines Configuration-Files zu erstellen.

Wie jede XML-Datei benötigt auch diese hier eine Kopfzeile, die man einfach immer wieder aus einer beliebigen anderen kopieren kann. Aber der Vollständigkeit halber wird sie hier noch mal in Listing 01 angegeben.

```
<?xml version="1.0"?>
```

Listing 01: Kopfzeile eines Configuration-Files

Die spitzen Klammern aus Listing 01 sind eine XML-Konvention, an die man sich gewöhnen muss. Das „xml“ bedeutet, dass dies ein XML-Dokument ist.
Nun folgt der eigentliche Inhalt einer XML-Datei. Dieser wird in XML immer als ein Baum repräsentiert. Wie jeder Baum benötigt auch ein XML-Dokument ein Wurzelelement. Dies wird hier immer „config“ genannt. Wie eben beschrieben, wird auch der Wurzelknoten mit solchen spitzen Klammern eingefasst und nennt sich von nun an ein öffnendes Tag. Jedes öffnende Tag stellt sozusagen einen Knoten in dem Baum dar. Damit die Verschachtelung der Knoten von einem Parser richtig verstanden werden kann, muss man zu jedem öffnenden auch ein schließendes Tag einfügen. Wenn man dies befolgt, sieht das Configuration-File nun wie in Listing 02 aus.

```
<?xml version="1.0"?>
<config>
</config>
```

Listing 02: Wurzeltag

Wie in Listing 02 zu erkennen ist, benötigt jedes schließende Tag einen Slash, um zu signalisieren, dass dies auch wirklich ein schließendes Tag ist.
Nachdem man nun den Wurzelknoten eingefügt hat, kann man beginnen sein eigentliches Netzwerk zu erstellen. Da XML-Dateien recht schnell unübersichtlich werden, sollte man sich schon im Voraus sein Netzwerk überlegt haben, um dieses dann möglichst geordnet und strukturiert in XML zu übersetzen. Eine handschriftliche Skizze ist oftmals sehr hilfreich. Als nächsten Schritt erstellt man erst einmal alle Tags für die Komponenten Switch und Host. Hat man dies erfolgreich erledigt, müsste das Configuration-File dasselbe Aussehen wie in Listing 03 aufweisen.

```
<?xml version="1.0"?>
<config>
<switch>
</switch>
<switch>
</switch>
<switch>
</switch>
<switch>
</switch>
<host>
</host>
<host>
</host>
<host>
</host>
<host>
</host>
</config>
```

Listing 03: Tags für Komponenten Switch und Host

Da alle Switches und Hosts gleichrangige Komponenten sind, werden sie auch alle ohne irgendeine Verschachtelung angelegt. Es müssen aber immer zuerst alle Switches und erst danach alle Hosts angelegt werden. Dies ist zwar XML egal, aber für den Parser des Programms ist dies von Bedeutung. Schon zu diesem sehr frühen Zeitpunkt kann man an Listing 03 sehen, dass ein Configuration-File ziemlich unübersichtlich wird. Deshalb ist ein geordnetes Vorgehen von entscheidender Wichtigkeit.
Nun fehlen aber noch die Ports. Da sie keine gleichrangigen Komponenten sind, sondern immer einem Switch zugeordnet sind, liegt es nahe, sie jeweils innerhalb des entsprechenden Tags anzulegen. Sie liegen dann eine Verschachtelungstiefe niedriger als die Switches, was auch sinnvoll ist. Da die Hosts keine weiteren Komponenten erhalten, kann man ihre Schreibweise etwas komprimieren, damit die Datei etwas übersichtlicher wird. In Listing 04 wurden bereits die Ports eingefügt und die Schreibweise der Hosts vereinfacht.

```
<?xml version="1.0"?>
<config>
<switch>
 <port/>
</switch>

<switch>
 <port/>
</switch>

<switch>
 <port/>
</switch>

<switch>
 <port/>
</switch>

 <host/>
 <host/>
 <host/>
 <host/>
</config>
```

Listing 04: Tags für Ports

Um die Übersichtlichkeit des Configuration-Files noch weiter zu erhöhen, wurden ein paar Leerzeilen, wie in Listing 04 zu sehen, eingefügt.
Nun sind erst einmal alle notwenigen Komponenten mittels Tags realisiert worden. Aber es fehlen noch die ganzen Attribute der einzelnen Komponenten. Diese könnte man theoretisch ebenfalls als Tags realisieren, aber jedem dürfte klar sein, dass dann niemand mehr durch solch eine Datei durchblickt. Deshalb hat man sich dazu entschieden, alle Attribute der Komponenten auch als Attribute in XML zu realisieren. Dies bedeutet, dass man sie einfach in das öffnende Tag jeder Komponente einfügt. Wie das genau auszusehen hat, kann man in Listing 05 sehen. Jeder der ein eigenes Configuration-File erstellen will, muss sich aber genau an die hier festgelegte Reihenfolge der Attribute halten.

```
<?xml version="1.0"?>
<config>
<switch id="001" mac_address="00:001" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2">
<port id="00001.01" priority="32.768"/>
<port id="00001.02" priority="32.768"/>
<port id="00001.04" priority="32.768"/>
</switch>

<switch id="002" mac_address="00:002" priority="32.768" port_number="4"
max_age="10" hello_time="2" forward_delay="2">
<port id="00002.01" priority="32.768"/>
<port id="00002.02" priority="32.768"/>
<port id="00002.03" priority="32.768"/>
<port id="00002.04" priority="32.768"/>
</switch>

<switch id="003" mac_address="00:003" priority="32.768" port_number="4"
max_age="10" hello_time="2" forward_delay="2">
<port id="00003.01" priority="32.768"/>
<port id="00003.02" priority="32.768"/>
<port id="00003.03" priority="32.768"/>
<port id="00003.04" priority="32.768"/>
</switch>

<switch id="004" mac_address="00:004" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2">
<port id="00004.01" priority="32.768"/>
<port id="00004.02" priority="32.768"/>
<port id="00004.04" priority="32.768"/>
</switch>

<host id="005" mac_address="01:001"/>
<host id="006" mac_address="01:002"/>
<host id="007" mac_address="01:003"/>
<host id="008" mac_address="01:004"/>

</config>
```

Listing 05: Einfache Attribute

Wie in Listing 05 zu erkennen, wurden mittlerweile alle einfachen Attribute, bei denen es keinerlei Probleme gibt, eingefügt. Bei XML benötigen Attribute immer zuerst einen Namen, gefolgt von dessen Wert in Anführungszeichen.
Nachdem dies nun geklärt ist, folgen nun die problematischeren Attribute wie die Positionen der Switches und Hosts. Der Anwender ist nicht gezwungen diese selber anzugeben. Sollte er sie weglassen, bestimmt das Programm für alle Komponenten selber eine Position, die man natürlich in keinster Weise als optimal bezeichnen kann. Aber man hat ja nach dem Laden die Möglichkeit, die einzelnen Komponenten zu verschieben und sich so manuell die ideale Position zu suchen. Möchte man dies nicht, kann man seine gewünschten Positionen schon in

dem Configuration-File angeben. Jedoch ist dies ziemlich schwierig, da man erst nach viel Übung mit dem Programm einigermaßen genau einschätzen kann, wo welche Koordinate liegt. In diesem Beispiel wurden jedoch die exakten Positionsangaben schon in dem Configuration-File gemacht, was man in Listing 06 sehen kann.

```
<?xml version="1.0"?>
<config>
<switch id="001" mac_address="00:001" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2" position_x="250" position_y="150">
<port id="00001.01" priority="32.768"/>
<port id="00001.02" priority="32.768"/>
<port id="00001.04" priority="32.768"/>
</switch>

<switch id="002" mac_address="00:002" priority="32.768" port_number="4"
max_age="10" hello_time="2" forward_delay="2" position_x="150" position_y="250">
<port id="00002.01" priority="32.768"/>
<port id="00002.02" priority="32.768"/>
<port id="00002.03" priority="32.768"/>
<port id="00002.04" priority="32.768"/>
</switch>

<switch id="003" mac_address="00:003" priority="32.768" port_number="4"
max_age="10" hello_time="2" forward_delay="2" position_x="450" position_y="250">
<port id="00003.01" priority="32.768"/>
<port id="00003.02" priority="32.768"/>
<port id="00003.03" priority="32.768"/>
<port id="00003.04" priority="32.768"/>
</switch>

<switch id="004" mac_address="00:004" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2" position_x="250" position_y="450">
<port id="00004.01" priority="32.768"/>
<port id="00004.02" priority="32.768"/>
<port id="00004.04" priority="32.768"/>
</switch>

<host id="005" mac_address="01:001" position_x="250" position_y="10"/>
<host id="006" mac_address="01:002" position_x="10" position_y="200"/>
<host id="007" mac_address="01:003" position_x="550" position_y="200"/>
<host id="008" mac_address="01:004" position_x="250" position_y="350"/>

</config>
```

Listing 06: Positionsangaben

Die nächste Schwierigkeit bilden jetzt die Verbindungsleitungen. Auch sie könnte man wieder als Tags realisieren, was hier aber nicht gemacht wurde. Alle Attribute also connected und path_costs der Wires wurden einfach zu den entsprechenden Ports hinzugefügt. Dies kann man in Listing 07 erkennen.

```
<?xml version="1.0"?>
<config>
<switch id="001" mac_address="00:001" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2" position_x="250" position_y="150">
<port id="00001.01" priority="32.768" connected="01001" path_costs="19"/>
<port id="00001.02" priority="32.768" connected="00003.01" path_costs="19"/>
<port id="00001.04" priority="32.768" connected="00002.01" path_costs="19"/>
</switch>

<switch id="002" mac_address="00:002" priority="32.768" port_number="4"
max_age="10" hello_time="2" forward_delay="2" position_x="150" position_y="250">
<port id="00002.01" priority="32.768" connected="00001.04" path_costs="19"/>
<port id="00002.02" priority="32.768" connected="00003.04" path_costs="19"/>
<port id="00002.03" priority="32.768" connected="00004.04" path_costs="19"/>
<port id="00002.04" priority="32.768" connected="01002" path_costs="19"/>
</switch>

<switch id="003" mac_address="00:003" priority="32.768" port_number="4"
max_age="10" hello_time="2" forward_delay="2" position_x="450" position_y="250">
<port id="00003.01" priority="32.768" connected="00001.02" path_costs="19"/>
<port id="00003.02" priority="32.768" connected="01003" path_costs="19"/>
<port id="00003.03" priority="32.768" connected="00004.02" path_costs="19"/>
<port id="00003.04" priority="32.768" connected="00002.02" path_costs="19"/>
</switch>

<switch id="004" mac_address="00:004" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2" position_x="250" position_y="450">
<port id="00004.01" priority="32.768" connected="01004" path_costs="19"/>
<port id="00004.02" priority="32.768" connected="00003.03" path_costs="19"/>
<port id="00004.04" priority="32.768" connected="00002.03" path_costs="19"/>
</switch>

<host id="005" mac_address="01:001" position_x="250" position_y="10"/>
<host id="006" mac_address="01:002" position_x="10" position_y="200"/>
<host id="007" mac_address="01:003" position_x="550" position_y="200"/>
<host id="008" mac_address="01:004" position_x="250" position_y="350"/>

</config>
```

Listing 07: Fertige Configuration-File 01

Listing 07 zeigt nun das fertige Configuration-File 01. Wenn die selbst erstellte Datei genauso aussieht, hat man die Erstellung erfolgreich abgeschlossen und kann sie nun ins Programm laden. Trotz der möglichst übersichtlichen Darstellung und der strukturierten Vorgehensweise ist es ziemlich aufwendig und schwierig, solch ein Configuration-File auf Anhieb komplett richtig aufzubauen. Es kann also schon etwas dauern und viel Geduld erfordern, bis das Netzwerk endlich korrekt im Programm dargestellt wird.

Der aufmerksame Leser wird sich jetzt noch daran erinnern, dass man auch Ports auf disabled setzen kann. Dazu verwendet man dieses mal einfach ein neues Tag. In Listing 08 wird nicht noch einmal das gesamte Configuration-File 01 gezeigt sondern nur ein Auszug, in dem einfach ein paar Ports von Switch 001 auf disabled gesetzt werden.

```
<switch id="001" mac_address="00:001" priority="32.768" port_number="3"
max_age="10" hello_time="2" forward_delay="2" position_x="250" position_y="150">
<port id="00001.01" priority="32.768" connected="01001" path_costs="19">
<disabled/>
</port>
<port id="00001.02" priority="32.768" connected="00003.01" path_costs="19">
<disabled/>
</port>
<port id="00001.04" priority="32.768" connected="00002.01" path_costs="19"/>
</switch>
```

Listing 08: Ports auf disabled gesetzt

6.3 Visualisierung eines Netzwerks

Nun hat man endlich sein eigenes Configuration-File erstellt und kann sie ins Programm laden. Dort wird das Netzwerk sofort im Bereich zur Darstellung es Netzwerks visualisiert. In Kapitel 2 auf Seite 3 hat man ja schon auf mehreren Screenshots ein visualisiertes Netzwerk gesehen. Jedoch wurde dort noch nicht auf alle Möglichkeiten, die der Nutzer hat, eingegangen. Zur Veranschaulichung folgt nun noch mal ein Screenshot mit dem visualisierten Netzwerk aus Configuration-File 01. Dieses kann man in Abbildung 27 sehen.

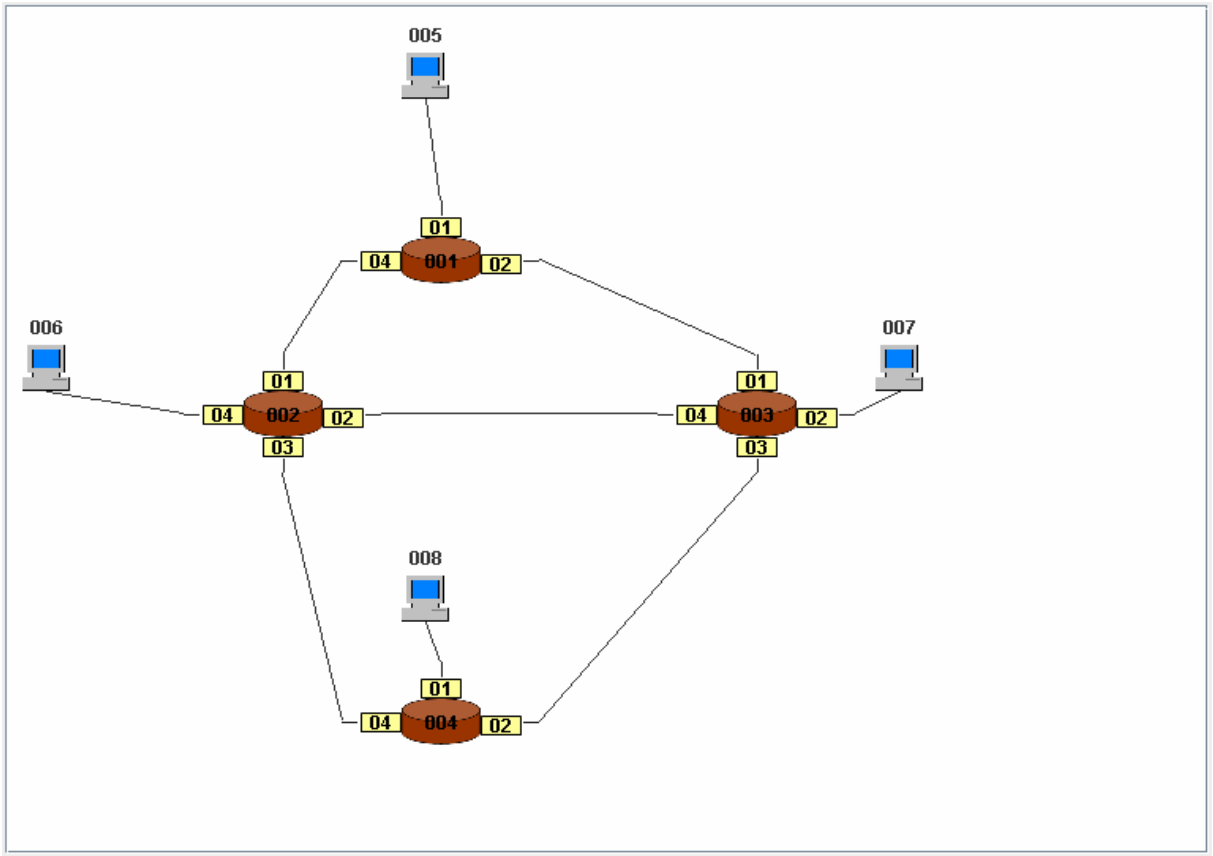

Abbildung 27: Visualisiertes Netzwerk nach dem Laden des Configuration-File 01

Wie die einzelnen Komponenten des Netzwerks visualisiert werden, wurde schon in Kapitel 2 auf Seite 3 beschrieben. So gut positioniert sind die Komponenten eines Netzwerks nur, wenn man die exakten Positionsangaben schon in dem Configuration-File angibt oder sie nach dem Laden manuell anordnet. Dies kann man einfach mittels Drag&Drop tun. Sollte der Platz im angezeigten Bereich einmal nicht für das komplette Netzwerk ausreichen, erscheinen automatisch Scrollbalken, die es einem ermöglichen, jeden Teil des Netzwerks auch weiterhin zu betrachten. Natürlich kann man auch manuell jede Komponente aus dem sichtbaren Bereich rausziehen.
Um einmal zu zeigen, wie dieses Netzwerk nach dem Laden aussieht, wenn man keine Positionsangaben angibt, folgt nun Abbildung 28.

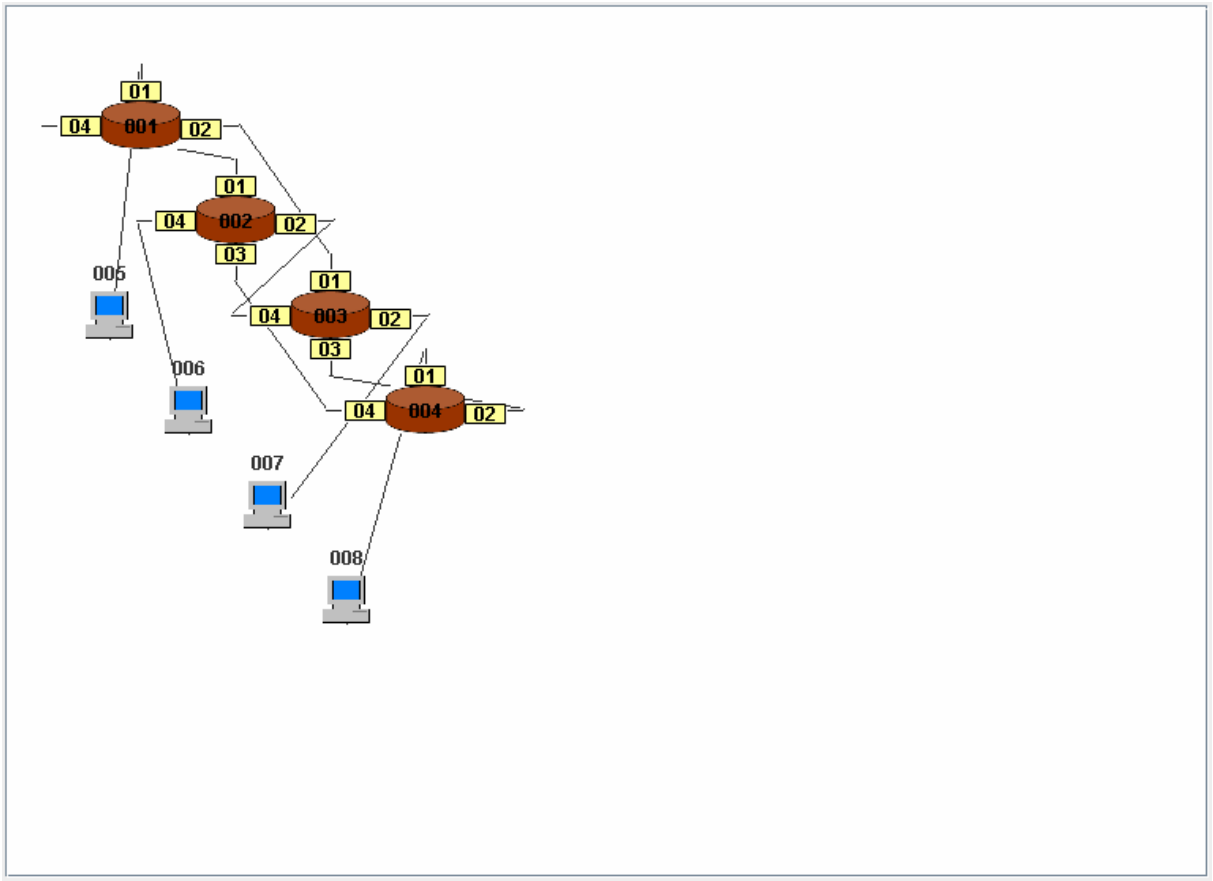

Abbildung 28: Netzwerk nach dem Laden des Configuration-File 01 ohne Positionsangaben

Man kann in Abbildung 28 gut erkennen, dass diese Darstellung des Netzwerks so nicht wirklich zu gebrauchen ist. Deshalb wird man die Komponenten des Netzwerks manuell positionieren müssen.

7 Implementation

In Kapitel 7 wird auf die Implementation des Programms eingegangen. Es wird zuerst ein Überblick über die Struktur des Programms mit all seinen Klassen gegeben. Anschließend wird erklärt, wie ein Configuration-File geparst und ungeparst wird, um dann zu beschreiben, wie die Visualisierung des Netzwerks implementiert wurde. Nachdem dann noch die Realisierung des Hauptfensters gezeigt wurde, wird endlich auf alle Bestandteile eines Netzwerks und den beiden Nachrichtentypen, also Switche, Ports, Hosts, Wire, BPDU und Message, eingegangen. Danach wird erklärt wie das Starten, Pausieren und Stoppen des Spanning Tree Algorithmus realisiert wurde, um am Ende noch auf die beiden Nebenfenster SAT-Mac Tables und Configurations einzugehen.

7.1 Programmstruktur

Hier wird erst einmal auf die grobe Programmstruktur eingegangen. Dazu folgt direkt die Abbildung 29, die einen Überblick über alle Klassen des Programms liefert.

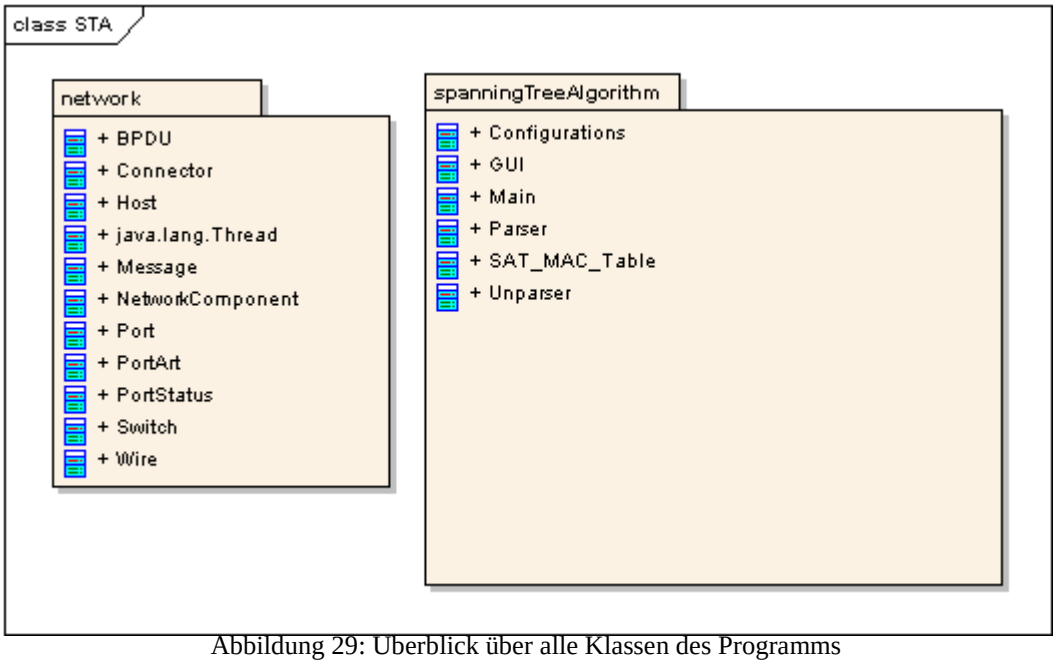

Abbildung 29: Überblick über alle Klassen des Programms

Wie in Abbildung 29 zu sehen, besteht das Programm aus zwei Paketen und zwar „network“ und „spanningTreeAlgorithm“. Im ersten sind alle Klassen enthalten, die die Bestandteile und Nachrichten eines Netzwerks repräsentieren. Das zweite beinhaltet alle Klassen, die für die graphische Benutzeroberfläche des Programms und für das Parsen bzw. Unparsen eines Configuration-Files verantwortlich sind. Da zwischen den Klassen des Pakets „spanningTreeAlgorithm“ keine relevanten Beziehungen vorhanden sind, ist hier keine Abbildung notwenig.
Das Einzige, was man vielleicht erwähnen sollte, ist, dass die Klasse GUI die wichtigste ist und das Hauptfenster mit allen Menüpunkten und Anzeigen implementiert. Da hier alle Schaltflächen angelegt wurden, werden von hier aus alle anderen Fenster wie Configurations aufgerufen. Auch werden von hier aus alle Threads gestartet und der Parser bzw. Unparser verwendet. Die Visualisierung des Netzwerks befindet sich ebenfalls in dieser Klasse.
Die restlichen Klassen dieses Pakets tragen eigentlich selbsterklärende Namen, die direkt verraten, was dort implementiert wurde. In der Klasse Parser befindet sich der Parser des Programms, der ein Configuration-File einliest und die Klasse Unparser enthält den Unparser,

der aus den Informationen innerhalb des Programms wieder eine XML-Datei erstellt. In Configurations wurden das Fenster, in dem man nachträglich noch Einstellungen an den Komponenten eines Netzwerks durchführen kann, implementiert und in SAT_Mac_Table findet man alle SAT-Mac Tables der Switche. Es lohnt sich nicht sonderlich, auf diese Klassen an dieser Stelle noch weiter einzugehen.
Im Paket „network“ befinden sich alle Klassen, die das Netzwerk beschreiben. Auch ihre Namen sind selbsterklärend, jedoch bestehen zwischen diesen Klassen wichtige Beziehungen, die hier näher erklärt werden sollten, was auch nun geschieht. Zu Beginn kommt erst einmal Abbildung 30, in der alle Klassen dieses Pakets mit ihren Beziehungen sowie den wichtigen Attributen und Methoden zu sehen sind.

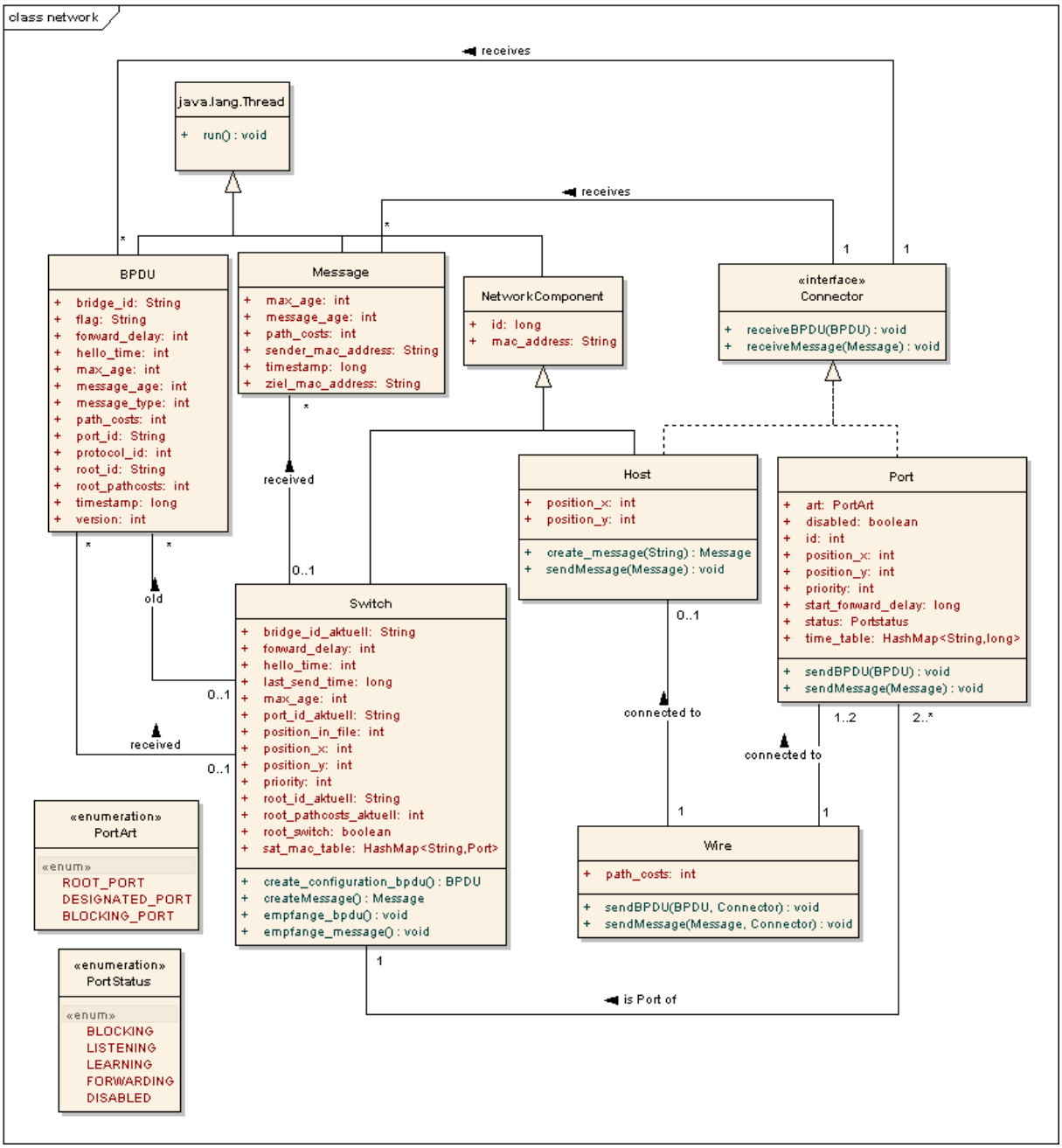

Abbildung 30: Das Paket „network“ mit allen Klassen

Wie man in Abbildung 30 erkennen kann, gibt es drei entscheidende Klassen und zwar Switch, Host und Port. Was sich dahinter verbirgt, dürfte wohl klar sein. Die ersten beiden haben die gemeinsame Oberklasse NetworkComponent, die selber von der Klasse Thread abgeleitet ist. Die Oberklasse wurde eingeführt, weil beide Unterklassen zwei gemeinsame Attribute verwenden und zwar die id und die mac_adresse. Für Host und Port wurde ein gemeinsames Interface erstellt, weil beide BPDUs und Messages empfangen können müssen. Wie man an der Klasse Wire erkennen kann, verbindet also eine Wire immer einen Port mit einem anderen oder mit einem Host. Was auch klar sein dürfte, ist, dass jeder Switch mindestens zwei Ports haben muss und jeder Port immer genau einem Switch zugeordnet ist. Dann gibt es noch die beiden Nachrichtentypen BPDU und Message, die auch von Thread abgeleitet sind. Jede der drei Hauptklassen kann beliebig viele Nachrichten beider Typen empfangen. Zuletzt ist noch zu erwähnen, dass es zwei Enumerations gibt und zwar Portart und Portstatus.
In der Klasse Switch findet man die gesamte Implementation des Spanning Tree Algorithmus.

7.2 Implementation des Parsers

Was ein Parser ist, dürfte wohl den meisten ungefähr klar sein. Er ist ein kleines Programm, das eine Datei in diesem Fall ein Configuration-File liest und sie in einen Syntaxbaum umwandelt. Er ermöglicht es an alle Informationen innerhalb einer Datei zu gelangen und sie weiter zu verwenden. Genau dies muss auch mit jedem Configuration-File geschehen, da man schließlich die Informationen aus dieser Datei braucht.
Da diese Dateien in XML erstellt werden und das Programm in Java geschrieben wurde, liegt es nahe StAX zum Parsen zu verwenden. StAX ist schon in Java integriert und stellt einen Kompromiss zwischen DOM und SAX dar. Des Weiteren wurde sich dazu entschieden die Dateien mit Hilfe des Cursor-Verfahrens einzulesen. Dabei iteriert der Parser mittels einer Tiefensuche über das gesamte Dokument und liefert eine Reihe von Events zurück. Im Anschluss wird geprüft, ob das zurück gelieferte Event nun ein START_ELEMENT oder ein END_DOCUMENT ist. Wenn letzteres zutrifft, wurde das Dokument vollständig iteriert und das Einlesen ist beendet. Das eigentliche Parsen findet natürlich in dem ersten Fall statt und endet erst, wenn das Ende des Dokuments erreicht wurde. [14]
Genau so wurde es auch im Programm umgesetzt. Es wird zuerst ein Parser erzeugt, der über das gesamte Dokument iteriert. Bei dieser Iteration geht der Cursor Schritt für Schritt durch das Dokument. Bei jedem Schritt wird geprüft, ob das Tag, auf dem sich der Cursor gerade befindet, ein Switch, Port oder Host ist. Dem aufmerksamen Leser wird auffallen, dass noch ein Tag fehlt und zwar das disabled-Tag. Natürlich wird auch darauf geprüft. Da in einem Configuration-File nur diese vier Tags vorkommen können, wird bei jedem Schritt des Cursors auch einer der Fälle zutreffen und dann wird eine entsprechende Methode aufgerufen, die nach den Attributen des Tags schaut. Dies alles kann man in Listing 09 erkennen.

```
public void parse(File file) throws Exception {
	try {
		FileInputStream in = new FileInputStream(file);
		XMLInputFactory factory = XMLInputFactory.newInstance();
		XMLStreamReader parser = factory.createXMLStreamReader(in);
		…
		while (true) {
			int event = parser.next();
			if (event == XMLStreamConstants.END_DOCUMENT) {
				parser.close();
				break;
			}
			if (event == XMLStreamConstants.START_ELEMENT) {
				// Einlesen von Switchs
				if (parser.getLocalName().toString().equals("switch")) {
					try {
						port_counter = 0;
						load_switch(parser);
					} catch (Exception e) {
						e.printStackTrace();
					}
				}
				// Einlesen von Ports
				else if (parser.getLocalName().toString().equals("port")) {
					try {
						load_port(parser);
					} catch (Exception e) {
					e.printStackTrace();
					}
				}
				…
			}
		}
			try {
				load_wire2();
			} catch (Exception e) {
				e.printStackTrace();
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (FactoryConfigurationError e) {
			e.printStackTrace();
		} catch (XMLStreamException e) {
			e.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
```

Listing 09: Parsen eines Configuration-Files

Wie an den Auslassungspunkten in Listing 09 zu sehen, wurde an zwei Stellen mehrere Zeilen Quellcode aus Platzgründen weggelassen. Diese spielen aber für das Verständnis keine Rolle. Im ersten Fall wurden einfach ein paar Variablen-Initialisierungen weggelassen, die in den folgenden Methoden benötigt werden. Im zweiten wurden die Prüfungen auf die Tags disabled und Host weggelassen, da sie identisch mit den beiden gezeigten sind. Wer den vollständigen Quellcode einsehen möchte, kann in die entsprechende Java-Datei schauen. Außerdem kann man in Listing 09 erkennen, dass die ganzen Prüfungen innerhalb einer Endlosschleife liegen und diese erst verlassen wird, wenn das Ende des Dokuments erreicht wurde. Die Methode load_wire2() wird nur einmal und zwar nach Verlassen der Schleife ausgeführt. Sie wird erst später in diesem Kapitel erklärt.
Zuerst wird jetzt die Methode load_switch() vorgestellt, der wie alle anderen Methoden das Parser-Objekt mitgegeben wird. Dort wird bei jedem Aufruf direkt zu Beginn ein Switch-Objekt angelegt. Anschließend werden alle Attribute, die im aktuellen Switch-Tag stehen eingelesen und dem Switch-Objekt übergeben. Sobald die Mac-Adresse eingelesen wurde, wird das Switch-Objekt in der HashMap switch_map, die in der Klasse GUI zu finden ist, mit der entsprechenden Mac-Adresse als Schlüssel gespeichert. Bei den Attributen position_x und postion_y wird vorher natürlich geprüft, ob diese überhaupt vorhanden sind. Nachdem diese beiden Attribute eingelesen wurden, wird unter Umständen das Panel, in dem das Netzwerk visualisiert werden soll, vergrößert. Gegen Ende der Methode wird für die Ports ein Array mit der Länge, die im Attribut port_number angegeben wurde, angelegt und ebenfalls dem Switch-Objekt übergeben. Als letztes wird dann die Methode visualize_Switch(), die sich in der Klasse GUI befindet, aufgerufen, um die Switch-Objekte zu visualisieren. Dieser Methode wird natürlich das Switch-Objekt mitgegeben. Dies ist in Listing 10 und 11 zu sehen.

```
public void load_switch(XMLStreamReader parser) {
        aswitch = new Switch();
        switch_anzahl = switch_anzahl + 1;

        // Lade id
        String id = parser.getAttributeValue(0);
        aswitch.id = Integer.parseInt(id);

        // Lade mac_address
        String mac_address = parser.getAttributeValue(1);
        if (mac_address.contains(":")) {
                mac_address = mac_address.replaceAll(":", "");
        }
        aswitch.mac_address = mac_address;
        GUI.switch_map.put(aswitch.mac_address, aswitch);
        switch_order.put(position_in_file, aswitch.mac_address);
        position_in_file = position_in_file + 1;
        …
```

Listing 10: Einlesen der Switch-Attribute 01

In Listing 10 sieht man, wie die ID und die Mac-Adresse eines Switch-Tags eingelesen und dem Switch-Objekt zugewiesen werden. Auch wird hier dieses Objekt in die HashMap switch_map eingefügt. Wie man an den Auslassungspunkten am Ende erkenne kann, ist die Methode hier noch nicht zu ende, sondern geht in Listing 11 weiter.

```
…
// Lade position_x
// Wenn keine X-Position vorhanden, wird Default-Wert genommen
if (parser.getAttributeCount() != 9) {
        if (switch_anzahl % 5 == 0) {
                aswitch.position_x = 5 * 60 + switch_neue_position_x;
                switch_neue_position_x = switch_neue_position_x + 150;
        } else {
                aswitch.position_x = (switch_anzahl % 5) * 60
                                + switch_neue_position_x;
        }
} else {
        String position_x = parser.getAttributeValue(7);
        aswitch.position_x = Integer.parseInt(position_x);
}

// Größe des Panels anpassen
double rechte_seite = GUI.panel_netzwerk.getBounds().getMaxX();
double untere_seite = GUI.panel_netzwerk.getBounds().getMaxY();
// Wenn Netzwerk über rechten Rand hinausgeht
if (aswitch.position_x + 80 >= rechte_seite) {
        GUI.panel_netzwerk.setPreferredSize(new Dimension(
                        aswitch.position_x + 100, (int) untere_seite));
}

// Wenn Netzwerk über unteren Rand hinausgeht
if (aswitch.position_y + 60 >= untere_seite) {
        GUI.panel_netzwerk.setPreferredSize(new Dimension(
                        (int) rechte_seite, aswitch.position_y + 100));
}
…
// Lade ports und Portnumber
int port_number = Integer.parseInt(parser.getAttributeValue(3));
aswitch.ports = new Port[port_number];

GUI.visualize_Switch(aswitch);
}
```

Listing 11: Einlesen der Switch-Attribute 02

Wie in Listing 11 zu sehen, geht hier die Methode load_switch weiter, jedoch wurde auch hier wieder ein Teil des Quellcodes weggelassen. Die Überprüfung, ob die beiden Attribute position_x und position_y in dem Configuration-File vorhanden sind, wird einfach mittels der Anzahl der Attribute des Switch-Tags durchgeführt. Da sie die beiden letzten Attribute sind, reicht diese Prüfung aus. Sollte die Anzahl nicht stimmen, wurden sie weggelassen und es muss die Position der Komponente automatisch bestimmt werden. Es werden immer fünf Switches leicht versetzt in einer diagonalen Reihe positioniert. Dann wird wieder oben mit einer neuen Reihe begonnen. Anschließend wird geprüft, ob eine der Switch-Komponenten außerhalb des sichtbaren Bereichs des Panels zur Darstellung des Netzwerks liegt. Sollte dies der Fall sein, wird das Panel vergrößert. Nun wurden wieder ein paar Codezeilen

weggelassen, in denen die letzten Switch-Attribute eingelesen werden. Zum Schluss wird dann die Methode zur Visualisierung aufgerufen.

Auf das Parsen der Host-Tags wird hier nicht eingegangen, da es vollkommen analog zum Parsen der Switch-Tags vonstatten geht. Auch die Host-Objekte werden in einer eigenen HashMap, die host_map heißt und sich auch in der Klasse GUI befindet, gespeichert. Die Mac-Adresse dient hier ebenfalls als Schlüssel.

Nun werden die Ports eines Switch-Tags geparst. Auch dies funktioniert sehr ähnlich dem der beiden vorherigen Komponenten. Jedoch wird hier nochmals kurz auf die entscheidenden Stellen eingegangen.

```
public void load_port(XMLStreamReader parser) throws Exception {
        if (port_counter >= aswitch.ports.length) {
                GUI.remove_Panel();
                JOptionPane.showMessageDialog(null, "There are too many Ports.");
                        throw new Exception("The are too many Ports.");
                } else {
                // Lege Ports an und füge sie ins Array ein
                Port aport = new Port();
                aswitch.ports[port_counter] = aport;
                …
                // Zuordnung von Ports zu Switchs
                aswitch.ports[port_counter].port_of_switch = aswitch;

                GUI.visualize_Port(aswitch.ports[port_counter],
                aswitch.ports[port_counter].id, aswitch.position_x,aswitch.position_y);

                if (!(wire_end.containsKey(id))) {
                        load_wire1(parser);
                }
                port_counter = port_counter + 1;
        }
}
```

Listing 12: Einlesen der Port-Attribute

Am Anfang von Listing 12 wird zuerst einmal geprüft, ob es mehr Ports gibt als im Attribut port_number angegeben wurde. Sollte dies nicht der Fall sein, wird wie üblich erst einmal bei jedem Aufruf dieser Methode ein Port-Objekt angelegt. Dann wird das Port-Objekt in das Array eines Switches eingefügt. Um später einfach von einem Port den dazugehörigen Switch ermitteln zu können, merkt sich jeder Port zu welchem Switch er gehört. Nun wird die Methode visualize_Port() aufgerufen, um das Port-Objekt zu visualisieren. Am Ende der Methode wird dann noch die Methode load_wire1() aufgerufen. Da die Wires in dem Configuration-File nicht als Tags realisiert wurden, müssen die Wire-Objekte manuell erzeugt werden. Dies geschieht in der eben genannten Methode. Die Wire-Objekte werden in zwei HashMaps gespeichert und zwar in wire_end und in wire_start. Als Schlüssel wird jeweils das Attribut connected genommen. Um doppelte Wire-Objekte zu verhindern, wird die Methode nur aufgerufen, wenn der Schlüssel noch nicht in der HashMap wire_end vorkommt.

Das Tag disabled wird hier auch nicht näher betrachtet, da es ziemlich einfach ist. Es wird einfach das Label des entsprechenden Ports deaktiviert, indem der Portstatus auf disabled gesetzt wird.
Nun wird die Methode load_wire1() näher betrachtet. Sie kann man in Listing 13 anschauen.

```
public void load_wire1(XMLStreamReader parser) throws Exception {
	// Lege Objekte wire an
	awire = new Wire();

	// Lade connected und füge in Hashmap ein
	String connected = parser.getAttributeValue(2);
	if (!wire_end.containsKey(connected)) {
		wire_end.put(connected, awire);
		// Zuordnung von Wire zum Port
		Port bport = aswitch.ports[port_counter];
		bport.wire = awire;
	} else {
		GUI.remove_Panel();
		throw new Exception("A port must not have more than one wire.");
	}

	// Lade pathcosts
	String pathcosts = parser.getAttributeValue(3);
	awire.path_costs = Integer.parseInt(pathcosts);

	// Startport bestimmen
	String port_id_teil1 = aswitch.mac_address;
	if (port_id_teil1.contains(":")) {
		port_id_teil1 = port_id_teil1.replace(":", "");
	}
	wire_start.put(port_id_teil1 + "_"
			+ Long.toString(aswitch.ports[port_counter].id), awire);
	awire.start = aswitch.ports[port_counter];
}
```

Listing 13: Einlesen der Wire-Attribute

Wie üblich wird auch in Listing 13 zuerst für die Komponente Wire ein Wire-Objekt angelegt. Anschließen wird das Attribut connected eingelesen, in dem steht, mit welchem Port bzw. mit welchem Host die Wire verbunden ist. Sollte es noch keinen Eintrag in der HashMap wire_end mit diesem Schlüssel geben, wird das Wire-Objekt eingefügt. Der Port merkt sich neben dem Switch, zu dem er gehört, auch die angeschlossene Wire. Mit dieser Abfrage verhindert man auch, dass an einem Port zwei Wires angeschlossen sein können. Zu den Pfadkosten braucht man nichts weiter zu sagen. Jedes Wire-Objekt merkt sich neben den Pfadkosten auch noch die Anfangs- und Endkomponente. Die Anfangskomponente ist immer ein Port, da diese Methode schließlich von einem Port aufgerufen wird. Jedoch die Endkomponente kann entweder ein Host oder ein Port sein. Da dies alles während dem Parsen geschieht, ist es wahrscheinlich, dass die Endkomponente noch gar nicht eingelesen wurde. Deshalb kann zu diesem Zeitpunkt nur der Startport bestimmt werden. Dies ist einfach der Port, der diese Methode aufgerufen hat.

Nun dürfte man auch erraten können, wozu die Methode load_wire2() dient. Sie bestimmt nämlich zu jeder Wire die zugehörige Endkomponente.

```
public void load_wire2() throws Exception {
	// Über alle Schlüssel iterieren
	for (String key : wire_end.keySet()) {
		String[] parts = key.split(Pattern.quote("."));
		// Falls awire mit anderem Port verbunden
		if (parts.length == 2) {
			awire = wire_end.get(key);
			Switch aswitch = GUI.switch_map.get(parts[0]);
			// Suche richtigen Port und füge ihn dem awire.end hinzu
			for (int i = 0; i < aswitch.ports.length; i++) {
				Port aport = aswitch.ports[i];
				String id_zwischen = new Integer(aport.id).toString();
				if (id_zwischen.length() == 1) {
					id_zwischen = "0" + id_zwischen;
				}
				if (id_zwischen.equals(parts[1])) {
					awire.end = aport;
				}
			}
		}
		// Falls awire mit host verbunden
		else if (parts.length == 1) {
			awire = wire_end.get(key);
			Host ahost = GUI.host_map.get(parts[0]);
			ahost.wire = awire;
			awire.end = ahost;
		}
		// Weise jedem Port die zugehörige Wire zu
		if (awire.end instanceof Port) {
			Port bport = (Port) awire.end;
			bport.wire = awire;
		}
		if (awire.start instanceof Port) {
			Port bport = (Port) awire.start;
			bport.wire = awire;
		}
	}
	…
```

Listing 14: Bestimmen der Endkomponenten 01

Listing 14 zeigt den ersten Teil der Methode load_wire2(). Es wird hier zuerst über die HashMap wire_end iteriert und geprüft, ob die eingetragene Endkomponente ein Port oder ein Host ist. Dies kann man einfach durch die Länge des Schlüssels rausbekommen. Sollte dieser aus zwei Teilen bestehen, war es eine ID von einem Port. Ansonsten war es die Mac-Adresse von einem Host. Sollte es ein Port gewesen sein, muss man sich erst einmal das entsprechende Port-Objekt holen, um dieses dann der Wire als Endkomponente zuweisen zu können. Sollte es jedoch ein Host gewesen sein, weist man halt den Host der Wire als

Endkomponente zu. Nun muss man nur noch dem Port die angeschlossene Wire zuweisen, was keine Schwierigkeit darstellen dürfte.
Im zweiten Teil dieser Methode, die in Listing 15 zu sehen ist, kommt zuerst ein Test, ob auch alle Wires eine Anfangs- und eine Endkomponente haben. Sollte dies nicht der Fall sein, wird der Nutzer darauf hingewiesen.

```
        // Überprüfung, ob Wire auch Anfangsport und Endport hat
        int host_counter = 0;
        for (String key1 : wire_end.keySet()) {
                awire = wire_end.get(key1);
                if (awire.end == null) {
                        GUI.remove_Panel();
                        throw new Exception("The port " + key1 + " does not exist.");
                } else if (awire.end instanceof Host) {
                        host_counter = host_counter + 1;
                }
        }
        if (host_counter != GUI.host_map.size()) {
                GUI.remove_Panel();
                throw new Exception("Some hosts are not correct connected.");
        }

        // Rufe für alle richtigen Wires Methode zum Zeichnen auf
        for (int i = 0; i < wire_end.size(); i++) {
                GUI.visualize_Wire();
        }
}
```

Listing 15: Bestimmen der Endkomponenten 02

Als letztes wird in Listing 15 die Methode visualize_Wire() der Klasse GUI aufgerufen, um die Verbindungslinien zeichnen zu lassen.
In diesem Unterkapitel wurde ausführlich erklärt wie der Parser im Rahmen dieser Bachelorarbeit implementiert wurde. Am sinnvollsten ist es, direkt im Anschluss den Unparser vorzustellen, was auch in Unterkapitel 7.3 geschieht.

7.3 Implementation des Unparsers

Da nun jeder weiß, was ein Parser ist, dürfte wohl nicht so schwer zu erraten sein, was ein Unparser ist. Er macht nämlich genau die umgekehrte Arbeit des Parsers. Er erstellt aus Informationen, die er von dem Programm erhält, eine Datei in diesem Fall natürlich ein Configuration-File. Auch dieses File muss wieder in XML erstellt werden und muss nahezu den identischen Aufbau haben wie die, die geparst wurde. Das bedeutet, dass die Reihenfolge der Komponenten und der Attribute beachtet werden muss. Die Werte dieser können und werden natürlich andere sein. Auch werden die Positionsangaben dieses Mal in jedem Fall in der Datei enthalten sein. Jedoch werden die Angaben zu den Wires nur noch einmal vorhanden sein und die neue Datei wird auch nicht mehr formatiert sein.

Wie zu Beginn des Parsers muss zuerst einmal ein Objekt erzeugt werden, dass die Aufgabe des Unparser übernimmt. Dies kann natürlich nicht wieder ein Parser-Objekt sein, sondern es ist ein writer.

Mit ihm wird zu allererst der Kopf einer jeden XML-Datei und direkt im Anschluss der Wurzelknoten config erstellt. Nun muss man alle Informationen, die man beim Parser den einzelnen Objekten zugewiesen hat, nacheinander anpacken und in das Configuration-File raus schreiben. Hier muss man genauso wie beim manuellen Anlegen solch eines Files Schritt für Schritt vorgehen. Da man beim Unparsen nicht einfach so an eine beliebige Stelle innerhalb des Dokuments springen kann, muss man sehr auf die Reihenfolge achten, in der man die Tags und die Attribute erstellt.
Nach dem öffnenden Wurzeltag muss natürlich zuerst einmal ein öffnendes Switch-Tag angelegt werden. Dort kommen dann alle Attribute des Switch-Objekts rein. Dies kann man in Listing 16 sehen.

```
public void unparse(String toSavePath) {
        XMLOutputFactory factory = XMLOutputFactory.newInstance();
        try {
                XMLStreamWriter writer = factory.createXMLStreamWriter(new
                        FileOutputStream(toSavePath));

                // Erzeuge Header
                writer.writeStartDocument();

                // Erzeuge Start-Tag Wurzelknoten
                writer.writeStartElement("config");

                String switch_mac;
                for (int i = 0; i < Parser.position_in_file; i++) {
                        switch_mac = Parser.switch_order.get(i);

                        // Erzeuge Start-Tag Switch
                        writer.writeStartElement("switch");

                        // Erzeuge Attribut id zu Switch
                        String id = new Long(GUI.switch_map.get(switch_mac).id)
                                        .toString();
                        if (id.length() == 1) {
                                id = "0" + "0" + id;
                        } else if (id.length() == 2) {
                                id = "0" + id;
                        }
                        writer.writeAttribute("id", id);
                        …
```

Listing 16: Unparsen der Switches

In Listing 16 wurde sich nur auf das erste Attribut ID beschränkt, da alle anderen analog funktionieren. Wie man an den Auslassungspunkten unten erkennt, geht die Methode hier noch weiter. Denn bevor man das schließende Switch-Tag setzen darf, müssen noch alle Port-Tags und eventuell disabled-Tags erzeugt werden. Dies kann man in Listing 17 sehen.

```
// Lese Ports ein
Port[] ports1 = GUI.switch_map.get(switch_mac).ports;
for (int j = 0; j < ports1.length; j++) {

        // Erzeuge Start-Tag Port
        writer.writeStartElement("port");
        …
        // Erzeuge Attribute Connected und Pfadkosten zu Port
        String key2 = port_id_teil1 + "_" + ports1[j].id;
        if (Parser.wire_start.containsKey(key2)) {
                Wire awire = Parser.wire_start.get(key2);
                // Falls Ende ein Host ist
                if (awire.end instanceof Host) {
                        Host host = (Host) awire.end;
                        String host_mac = host.mac_address;
                        writer.writeAttribute("connected", host_mac);
                }
                // Falls Ende eine anderer Port ist
                else {
                        Port port_ende = (Port) awire.end;
                        Switch switch_ende = port_ende.port_of_switch;
                        String switch_ende_mac = switch_ende.mac_address;
                        String port_ende_id;
                        if (new Integer(port_ende.id).toString().length() == 1) {
                                port_ende_id = switch_ende_mac + "." + "0"
                                                                + port_ende.id;
                                writer.writeAttribute("connected", port_ende_id);
                        } else {
                                port_ende_id = switch_ende_mac + "."
                                        + new Integer(port_ende.id).toString();
                                writer.writeAttribute("connected", port_ende_id);
                        }

                }
                writer.writeAttribute("pathcosts", new Integer(
                                                        awire.path_costs).toString());
}
```

Listing 17: Unparsen der Ports

Nachdem das öffnende Port-Tag erzeugt wurde, folgen eine ganze Reihe von Attributen, die in Listing 17 weggelassen wurden. Das einzige interessante Attribut ist connected, da es recht schwierig ist, an alle Informationen zu gelangen. Zuerst muss man die Wire, die an diesem Port angeschlossen ist, in der HashMap wire_start suchen. Wenn man diese gefunden hat, prüft man, ob das Ende dieser Leitung ein Port oder ein Host ist. Sollte der zweite Fall zutreffen, kann man direkt die Mac-Adresse ausgeben lassen. Sollte jedoch der erste Fall eintreten, muss man sich die ID des Ports erst einmal zusammenbauen. Dazu benötigt man die Mac-Adresse des Switches, zu dem der Endport gehört und die ID des Endports. Wenn man beide Informationen hat, kann man auch in diesem Fall das Attribut connected erzeugen lassen.
Falls der Port noch ein disabled-Tag enthalten sollte, muss man diesen analog zu den anderen anlegen, um dann endlich das schließende Port-Tag zu setzen. Wenn man mit allen Ports des

Switches fertig ist, kann man auch zu diesem das schließende Tag setzen. Nachdem man alle Switches abgeschlossen hat, kommen die Hosts dran, die aber genauso funktionieren. Ganz am Ende muss man noch das Wurzeltag schließen, das Dokument beenden und den writer schließen. Dies kann man in Listing 18 erkennen.

```
// Erzeuge End-Tag von Wurzelelement
writer.writeEndElement();
// Erzeuge Ende von Dokument
writer.writeEndDocument();
// Schließe Writer
writer.close();
```

Listing 18: Abschließen des Unparsers

Nun wurde auch die Implementation des Unparser beschrieben. Als nächster logischer Schritt kommt jetzt die Implementation der Visualisierung eines Netzwerks.

7.4 Implementation der Visualisierung eines Netzwerks

In diesem Unterkapitel wird erklärt, wie die Visualisierung eines Netzwerks implementiert wurde. Die Komponenten Switch, Port und Host sind mittels einfacher Labels realisiert worden, in die ein Bild für die entsprechende Komponente eingefügt wurde. Zur Positionierung der Labels verwendet man die Positionsangaben, die die Objekte besitzen. Die ID der jeweiligen Komponente ist natürlich kein Bild, sondern wird einfach mit der normalen setText() Methode eines Labels erzeugt. Da die Label alle automatisch erzeugt werden und somit alle gleich heißen, ist es problematisch später darauf nochmals zuzugreifen. Deshalb wird jedes Label dem entsprechenden Objekt übergeben, damit man immer direkten Zugriff auf dieses hat. Dies alles ist nicht wirklich schwierig und wird darum auch nicht näher erklärt.

Da man die Switches und Hosts verschieben können muss, benötigen diese Labels einen MouseListener. Sie sollen aber erst am gewünschten Ort auftauchen, wenn man die Maustaste loslässt. Eine Einschränkung gibt es jedoch. Man kann die Komponenten nur verschieben, solange der Spanning Tree Algorithmus nicht läuft, da dieser einfach zu viele Systemressourcen schluckt.
Das MouseEvent hat zwar die Methoden getX() und getY(), jedoch liefern sie nicht die aktuelle Position des Mauszeigers zurück. Durch sie bekommt man nur die Veränderung der X- bzw. Y-Koordinaten und muss diesen Wert dann auf die alte Positionsangabe aufaddieren.
Da man auch in der Lage sein soll eine Komponente aus den sichtbaren Bereich des Panels zu ziehen, müssen Abfragen implementiert werden, die überprüfen, ob sich der Mauszeiger außerhalb dieses Bereichs befindet. Wenn er sich rechts oder unterhalb vom Panel befindet, wird seine Größe einfach angepasst und Scrollbalken erscheinen. Dies klappt aber nicht, wenn er sich oberhalb oder links davon befindet. Sollte man ein Label dort rausziehen, bleibt es unerreichbar. Deshalb wird in diesen beiden Fällen das entsprechende Label einfach ganz links bzw. ganz oben am Rand positioniert. Man ist also nicht mehr in der Lage das Label oben oder links aus dem Panel rauszuziehen. In Listing 19 erkennt man das Verschieben eines Labels der Komponente Switch.

```
// Verschieben der Switche
label_Switch.addMouseListener(new MouseAdapter() {
	@Override
	public void mouseReleased(MouseEvent e) {
		if (stop_Simulation_Button.isEnabled() == false) {
			int position_x = e.getX();
			int position_y = e.getY();
			// Wenn Mauszeiger linken Rand überschreitet
			if (aswitch.position_x + position_x < panel_netzwerk
					.getBounds().getMinX() + 45) {
				aswitch.position_x = 45;
			}
			// Normales Verschieben innerhalb der Zeichenflache
			else {
				aswitch.position_x = aswitch.position_x + position_x- 25;
			}
			// Wenn Mauszeiger oberen Rand überschreitet
			if (aswitch.position_y + position_y < panel_netzwerk
					.getBounds().getMinY() + 30) {
				aswitch.position_y = 30;
			}
			// Normales Verschieben innerhalb der Zeichenflache
			else {
				aswitch.position_y = aswitch.position_y + position_y- 15;
			}
			// Größe des Panels anpassen
			double rechte_seite = panel_netzwerk.getBounds().getMaxX();
			double untere_seite = panel_netzwerk.getBounds().getMaxY();
			// Wenn Netzwerk über rechten Rand hinausgeht
			if (aswitch.position_x + 40 >= rechte_seite) {
				panel_netzwerk.setPreferredSize(new Dimension(
					aswitch.position_x + 100, (int) untere_seite));
			}

			// Wenn Netzwerk über unteren Rand hinausgeht
			if (aswitch.position_y + 60 >= untere_seite) {
				panel_netzwerk.setPreferredSize(new Dimension(
					(int) rechte_seite, aswitch.position_y + 100));
			}
			…
```

Listing 19: Verschieben eines Switch-Labels

Wie in Listing 19 zu sehen, geht die Methode noch weiter. Hier wurde erst einmal das Verschieben eines Labels gezeigt. Jedoch sind dadurch ein paar Probleme aufgetaucht, deren Lösung im Folgenden gezeigt wird.

Eines der Probleme war, dass nach dem Verschieben eines Labels dieses zweimal zu sehen war und zwar an der alten und an der neuen Stelle. Dieses wurde einfach umgangen, indem man alle alten Labels vom Panel gelöscht und sie dann neu erzeugt hat. Dies ist zwar etwas aufwendig, aber so konnte man das Problem auf einfache Weise lösen.
Das Verschieben der Labels führte auch dazu, dass die Schriftfarbe, die durch den Algorithmus gesetzt wurde, immer wieder zurück auf schwarz gewechselt ist. Also muss man nach dem Verschieben eines Labels dessen Schrift erneut einfärben.
Diese Probleme sieht man in Listing 20.

```
        // Löschen der alten Labels
        Component[] component_array = panel_netzwerk.getComponents();
        for (int i = 0; i < component_array.length; i++) {
                if (component_array[i].getName().equals("label_Port")) {
                        panel_netzwerk.remove(component_array[i]);
                } else if (component_array[i].getName().equals("label_Line")) {
                        panel_netzwerk.remove(component_array[i]);
                } else if (component_array[i].getName().equals("label_Switch")) {
                        panel_netzwerk.remove(component_array[i]);
                }
        }
        panel_netzwerk.updateUI();

        // Setzten der neuen Labels
        for (String id : switch_map.keySet()) {
        visualize_Switch(switch_map.get(id));
        for (int i = 0; i < switch_map.get(id).ports.length; i++) {
                visualize_Port(switch_map.get(id).ports[i], switch_map.get(id).ports[i].id,
                        switch_map.get(id).position_x, switch_map.get(id).position_y);
        }
}
// Neues zeichnen der Linien
visualize_Wire();
}
}
});
```

Listing 20: Probleme durch das Verschieben

Die Hosts werden auf genau dieselbe Weise visualisiert und verschoben wie die Switches. Deshalb wird hier nicht näher darauf eingegangen. Auch die Ports werden im Prinzip ähnlich dargestellt. Jedoch gibt es hier ein paar kleine erwähnenswerte Unterschiede. Es gibt, wie vorher schon gesehen, nur vier mögliche Positionen von Ports und zwar oben, rechts, unten und links von einem Switch. Je nachdem welche ID der Port hat, bekommt er eine der vier Positionen zugeteilt. Dies bedeutet, dass wenn ein Switch mehr als vier Ports hat, nur die ersten vier zu sehen sind, da die anderen genau unter den ersten vier liegen. Dies führt bei der Betrachtung der Wires vielleicht zu Verwirrung, weil es so aussieht, als wären an einem Port mehrere Wires angeschlossen, was aber nicht der Fall ist. Dies ist eine kleine Unschönheit, mit der man aber leben kann. Die Bestimmung, welche der vier möglichen Positionen ein Port letztendlich bekommt, erfolgt mittels Modulo-Rechnung. Wie dies funktioniert, wird hier als bekannt vorausgesetzt. Die Bestimmung der Positionen der Ports kann man in Listing 21 sehen.

```
// Bestimme Position der Ports
// Port links
if (id % 4 == 0) {
	label_Port.setLocation(position_x - 28, position_y + 8);
	label_Line.setSize(10, 3);
	label_Line.setLocation(position_x - 38, position_y + 14);
	aport.position_x = position_x - 38;
	aport.position_y = position_y + 14;
	icon1 = new ImageIcon("Linie_Waagerecht.png");
	bild3 = ((ImageIcon) icon1).getImage();
	bild4 = bild3.getScaledInstance(10, 3, 0);
}
// Port oben
else if (id % 4 == 1) {
	label_Port.setLocation(position_x + 25 - 15, position_y - 14);
	label_Line.setSize(3, 10);
	label_Line.setLocation(position_x + 24, position_y - 24);
	aport.position_x = position_x + 24;
	aport.position_y = position_y - 24;
	icon1 = new ImageIcon("Linie_Senkrecht.png");
	bild3 = ((ImageIcon) icon1).getImage();
	bild4 = bild3.getScaledInstance(3, 10, 0);
}
// Port rechts
else if (id % 4 == 2) {
	label_Port.setLocation(position_x + 48, position_y + 10);
	label_Line.setSize(10, 3);
	label_Line.setLocation(position_x + 77, position_y + 14);
	aport.position_x = position_x + 87;
	aport.position_y = position_y + 14;
	icon1 = new ImageIcon("Linie_Waagerecht.png");
	bild3 = ((ImageIcon) icon1).getImage();
	bild4 = bild3.getScaledInstance(10, 3, 0);
}
// Port unten
else if (id % 4 == 3) {
	label_Port.setLocation(position_x + 25 - 15, position_y + 29);
	label_Line.setSize(3, 10);
	label_Line.setLocation(position_x + 24, position_y + 43);
	aport.position_x = position_x + 24;
	aport.position_y = position_y + 53;
	icon1 = new ImageIcon("Linie_Senkrecht.png");
	bild3 = ((ImageIcon) icon1).getImage();
	bild4 = bild3.getScaledInstance(3, 10, 0);
}
```

Listing 21: Bestimmung der Positionen der Ports

Wenn man sich Listing 21 etwas genauer anschaut, fällt einem auf, dass es dort einige Codezeilen gibt, wo etwas von Bildern mit einer Linie zu lesen ist. Diese Bilder mit den Linien werden in extra Label eingefügt, die direkt an die der Ports angrenzen. Diese dienen

einfach dazu, dass der Übergang von der Wire zum Port etwas besser aussieht. Sie erfüllen aber keinerlei Aufgabe.
Nun wurde schon mehrmals von den Wires gesprochen, aber noch nicht erklärt, wie diese visualisiert werden. Dies geschieht nicht mit Labels, sondern sie werden richtig im Panel gezeichnet. Hierzu wurde die Klasse MyPanel implementiert, die von JPanel abgeleitet ist. Von dieser Klasse wurde dann die Methode paintComponent überschrieben, in der dann angegeben ist, wo und wie die Wires zu zeichnen sind. Dies Klasse kann man in Listing 22 sehen.

```
class MyPanel extends JPanel {
	…
	@Override
	protected void paintComponent(Graphics g) {
		super.paintComponent(g);
		if (wire_test == true) {
			for (String key : parse_object.wire_end.keySet()) {
				Port start_port = (Port) parse_object.wire_end.get(key).start;
				Connector end = parse_object.wire_end.get(key).end;
				// Wenn Ende = Port
				if (end instanceof Port) {
					Port end_port = (Port) end;
					g.drawLine(start_port.position_x,
						start_port.position_y, end_port.position_x,
						end_port.position_y);
				}
				// Wenn Ende gleich Host
				else if (end instanceof Host) {
					Host end_host = (Host) end;
					g.drawLine(start_port.position_x,
						start_port.position_y,
						end_host.position_x + 15,
						end_host.position_y + 50);
				}
			}
		}
	}
}
```

Listing 22: Visualisieren der Wires

Zu Beginn von Listing 22 wird zuerst einmal überprüft, ob das Configuration-File vollständig geparst wurde. Diese Bestätigung wird in der Methode visualize_Wire() gegeben. Sollte der Parser fertig sein, wird über die HasMap wire_end iteriert und die Start- und End-Komponenten jeder Wire bestimmt. Dann wird zwischen diesen beiden Komponenten eine Linie gezeichnet, die eine Wire repräsentieren soll.

7.5 Implementation des Hauptfensters

Zur Implementation des Hauptfensters gibt es nicht viel zusagen, da der Aufbau bereits ausführlich erklärt wurde und die Komponenten alle einfach ohne irgendwelche Schwierigkeiten zu implementieren sind. Das Einzige, das vielleicht erwähnenswert wäre, ist,

dass dieses Fenster als eines der wenigen ein Layout verwendet und zwar das SpringLayout. Dieses ermöglicht es, dass sich zumindest das Panel zur Darstellung des Netzwerks an die Größe des Fensters automatisch anpasst. Das Panel zur Darstellung des Algorithmus tut dies nur in vertikaler Richtung. Wie genau dieses Layout funktioniert und wie es umgesetzt wird, wird ebenfalls als bekannt vorausgesetzt.

7.6 Implementation der Switches

Zur Implementation der Switches gibt es schon mehr zu berichten. Wie schon vorher erwähnt, wurden die Switches mittels Threads realisiert, um die Arbeitsweise eines realen Netzwerks zu simulieren. Dies bedeutet, dass alle Switches parallel arbeiten und über kein globales Wissen vom Netzwerk verfügen. Sie erhalten nur durch die BPDUs Wissen über anderen Komponenten. Was dies für den Spanning Tree Algorithmus bedeutet, wurde schon in Kapitel 2 auf Seite 3 mehr als ausführlich beschrieben.
Aber wie arbeiten die implementierten Switches nun wirklich? Wie jeder Thread müssen auch diese erst einmal gestartet werden. Dies geschieht sinnvoller Weise mittels des Startbefehls des Algorithmus. Diesen kann man entweder über den Menüpunkt oder über den entsprechenden Button geben. Dann beginnen die Threads zu arbeiten und führen ihre run-Methoden aus. Dort warten sie erst einmal solange, bis alle Threads gestartet wurden. Erst dann gehen sie im Code weiter. Da der Thread nach einmaligen Durchlaufen des Quellcodes beendet wäre, wird der gesamte Code der run-Methode in eine Schleife gepackt, die solange läuft, solange der Algorithmus noch nicht beendet wurde. Dazu wird einfach geprüft, ob der Beenden-Button anklickbar ist oder nicht.
Als ersten Schritt werden alle Ports mit der entsprechenden Schriftfarbe eingefärbt. Dann wird geprüft, ob der Switch, der Root-Switch ist. Da dies beim ersten Durchlauf auf alle zutrifft, durchlaufen alle Ports die verschiedenen Stati, bis sie im Forwarding-Status sind. Dies kann man in Listing 23 sehen. Die Port-Stati und Port-Arten sind im Programm alle mit Enumerations realisiert worden.
Damit ein Port in den Folgestatus wechseln kann, muss zuerst ein Timer abgelaufen sein. Sobald sich alle Ports, die in den Folgestatus wechseln wollen, den Startzeitpunkt ihres Timers gemerkt haben, legt sich der entsprechende Switch-Thread mittels sleep-Befehl schlafen. Um sicherzustellen, dass jeder Port nur dann in den Folgestatus übergeht, wenn sein Timer abgelaufen ist, wird dies nach dem Aufwachen des Threads mit Hilfe des gespeicherten Startzeitpunktes kontrolliert.
Wer sich Listing 23 etwas näher anschaut, bemerkt, dass sich immer direkt vor und nach dem sleep-Befehl eine Abfrage befindet. Diese Abfrage spielt nur für das Pausieren des Algorithmus eine Rolle. Da ein Thread, der gerade schläft nicht den wait-Befehl ausführen kann, wurden diese Abfragen eingeführt. Sobald man auf den Pause-Button klickt, merkt der Thread, dass dieser Button geklickt wurde und deshalb prüft er immer vor und nach dem Schlafen, ob er warten soll.
In Listing 23 wird nur gezeigt, wie ein Switch-Thread seine Ports in den Listening- und anschließend in den Learning-Status setzt. Der Wechsel vom Learning- in den Forwarding-Status funktioniert genauso.

```
// Gehe zuerst in den Listening-Status
for (int i = 0; i < ports.length; i++) {
        if (ports[i].status == Portstatus.BLOCKING
                                        && ports[i].art != Portart.BLOCKING_PORT) {
                ports[i].status = Portstatus.LISTENING;
                ports[i].start_forward_delay = System.currentTimeMillis();
        }
}
// Switch-Thread soll sich schlafen legen
try {
        // Wenn Pause aktiviert, soll Thread warten
        if (should_wait == true) {
                synchronized (this) {
                        wait();
                }
        }
        sleep(forward_delay * 1000);
        // Wenn Pause aktiviert, soll Thread warten
        if (should_wait == true) {
                synchronized (this) {
                        wait();
                }
        }
} catch (InterruptedException e) {
        e.printStackTrace();
}
// Sobald Timer abgelaufen ist, gehe in den Learning-Status
for (int i = 0; i < ports.length; i++) {
        if (System.currentTimeMillis() >= ports[i].start_forward_delay
                                                + forward_delay * 1000) {
                if (ports[i].status == Portstatus.LISTENING) {
                        ports[i].status = Portstatus.LEARNING;
                        // Merke alle notwendigen Daten
                        String id_switch = new Long(id).toString();
                        if (id_switch.length() == 1) {
                                id_switch = "0" + "0" + id_switch;
                        } else if (id_switch.length() == 2) {
                                id_switch = "0" + id_switch;
                        }
                        root_id_aktuell = priority + "." + id_switch+ "." + mac_address;
                        label_root_id.setText(root_id_aktuell);
                        root_pathcosts_aktuell = 0;
                        label_root_pathcosts.setText(new Integer(
                                                root_pathcosts_aktuell).toString());
                        ports[i].start_forward_delay = System.currentTimeMillis();
                }
        }
}
…
```

Listing 23: Übergang eines Switches vom Listening- in den Learning-Status

Wie in Listing 23 zu sehen ist, bestimmt der Thread die für den Algorithmus notwendigen Informationen wie Root-Pfadkosten und Root ID. Ebenfalls sorgt er dafür, dass im Panel zur Anzeige des Algorithmus die entsprechenden Angaben erscheinen.

Sobald ein Port im Forwarding-Status ist, werden die ersten BPDUs erzeugt und über alle Ports, die sich in dem eben genannten Status befinden, verschickt. Dazu wird zuerst die Methode create_configuration_bpdu(), die eine BPDU erzeugt und dann die Methode sendBPDU(), die die BPDU verschickt, aufgerufen. Die erste der beiden Methoden, befindet sich in dieser Klasse und die zweite in der Klasse Port. Die Arbeitsweise des Erzeugens und Verschickens von BPDUs bei einem Root-Switch kann man im Listening 24 sehen.

```
// Nur wenn Hello-Time vestrichen ist, soll eine neue BPDU
// erzeugt und verschickt werden.
if (System.currentTimeMillis() >= last_send_time + hello_time* 1000) {
        // Erzeugen und Verschicken einer BPDU
        BPDU bpdu1;
        for (int i = 0; i < ports.length; i++) {
                if (ports[i].status == Portstatus.FORWARDING) {
                        Port bport = ports[i];
                        bpdu1 = create_configuration_bpdu();
                        String id_port = new Integer(bport.id).toString();
                        if (id_port.length() == 1) {
                                id_port = "0" + id_port;
                        }
                        bpdu1.port_id = mac_address + "." + id_port;
                        bpdu1.flag = "0";
                        bport.sendBPDU(bpdu1);
                        last_send_time = System.currentTimeMillis();
                }
        }
}
```

Listing 24: Erzeugen und Verschicken der ersten BPDUs

Aus was für Teilen eine BPDU besteht und was diese bedeuten, wurde bereits in Kapitel 2 auf Seite 3 ausführlich erklärt. Da hier die Methode create_configuration_bpdu() verwendet wurde, wird diese als nächstes beschrieben. Hier wird ein BPDU-Objekt angelegt, dem dann alle notwendigen Informationen mitgegeben werden. Am Ende von Listing 25 liefert die Methode dann dieses Objekt zurück. Bei der Zuweisung der Daten selbst gibt es keinerlei Schwierigkeiten. Jedes dieser BPDU-Objekte ist selber auch ein Thread, der aber hier noch nicht gestartet wird. Er wird einfach an den entsprechenden Port übergeben, der dann den Thread an die Wire weiterleitet.

```
public BPDU create_configuration_bpdu() {
	BPDU bpdu1 = new BPDU();
	bpdu1.root_id = root_id_aktuell;
	bpdu1.root_pathcosts = root_pathcosts_aktuell;
	String id_switch2 = new Long(id).toString();
	if (id_switch2.length() == 1) {
		id_switch2 = "0" + "0" + id_switch2;
	} else if (id_switch2.length() == 2) {
		id_switch2 = "0" + id_switch2;
	}
	bpdu1.bridge_id = priority + "." + id_switch2 + "." + mac_address;
	if (root_switch == true) {
		bpdu1.message_age = 1;
	}
	bpdu1.max_age = max_age;
	bpdu1.hello_time = hello_time;
	bpdu1.forward_delay = forward_delay;
	return bpdu1;
}
```

Listing 25: Erzeugen einer BPDU

Nun wurden also von allen Switches, die noch glauben, dass sie der Root-Switch sind, BPDUs verschickt. Irgendwann werden sie dann beim nächsten Switch ankommen. Dort nimmt der Port die Nachricht entgegen und reiht sie in die PriorityBlockingQueue inbox1 des Switches ein. Sortiert werden sie dort nach der Empfangszeit. Da der Switch-Thread seinen Quellcode immer wieder durchläuft, kommt er irgendwann wieder an die Stelle, wo er prüft, ob in inbox1 eine BPDU liegt. Dies kann man in Listing 26 sehen.

```
// Empfangen von BPDUs
if (!inbox1.isEmpty()) {
	try {
		bpdu2 = inbox1.take();
		// empfangene BPDU speichern
		synchronized (inbox2) {
			inbox2.add(bpdu2);
		}
		aport = (Port) bpdu2.empfaenger;
		aport.status = Portstatus.LISTENING;
		empfange_bpdu();
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}
```

Listing 26: Prüfung, ob BPDU angekommen ist

Wie in Listing 26 zu sehen, wird die erste BPDU nach erfolgreicher Prüfung aus inbox1 geholt und dabei gleichzeitig dort gelöscht. Da man die BPDUs auch nachher noch einsehen und anzeigen lassen möchte, wird sie sofort in eine ArrayList inbox2 eingefügt, wo sie dauerhaft gespeichert bleibt. Danach wird der Port bestimmt, wo die Nachricht empfangen

wurde und dieser wird sofort in den Listening-Status gesetzt. Zum Schluss wird dann die Methode empfange_bpdu() aufgerufen, die die BPDU auswertet.
Diese Methode ist nun an der Reihe betrachtet zu werden. Dies ist die entscheidendste Methode in der gesamten Klasse, da sich hier der gesamte Spanning Tree Algorithmus mit all seinen Prüfungen befindet. Da dieser Algorithmus in allen Einzelheiten schon in Kapitel 2 auf Seite 3 beschrieben wurde, wird hier nicht mehr alles gezeigt. Alle für den Algorithmus notwendigen Prüfungen wurden eins zu eins wie beschrieben umgesetzt. Deshalb werden diese nur exemplarisch vorgestellt. Zuerst wird der Root-Switch bestimmt. Dazu werden alle drei Teile der Root ID verglichen. Die Prüfung vom ersten dieser drei Teile sieht man in Listing 27.

```
public void empfange_bpdu() {
        String[] part1 = bpdu2.root_id.split(Pattern.quote("."));
        String[] part2 = root_id_aktuell.split(Pattern.quote("."));
        // In BPDU ist ein Root-Switch genannt, der kleinere Priorität hat als
        // der bisher angenommene Root-Switch
        if (Integer.parseInt(part1[0]) < Integer.parseInt(part2[0])) {
                root_switch = false;
                label_switch.setForeground(Color.BLUE);
                aport.root_pathcosts_zwischen = bpdu2.root_pathcosts
                                + aport.wire.path_costs;
                root_pathcosts_aktuell = aport.root_pathcosts_zwischen;
                aport.root_id_zwischen = bpdu2.root_id;
                root_id_aktuell = bpdu2.root_id;
                bridge_id_aktuell = bpdu2.bridge_id;
                port_id_aktuell = bpdu2.port_id;

                // Root-Port festlegen
                if (test_root_port == false) {
                        test_root_port = true;
                        aport.art = Portart.ROOT_PORT;
                        aport.label_port.setForeground(Color.GREEN);
                } else {
                        // Setze alle anderen Ports auf Designated
                        for (int i = 0; i < ports.length; i++) {
                                if (ports[i].id != aport.id) {
                                        ports[i].art = Portart.DESIGNATED_PORT;
                                        ports[i].label_port.setForeground(Color.BLUE);
                                }
                        }
                        aport.art = Portart.ROOT_PORT;
                        aport.label_port.setForeground(Color.GREEN);
                }
        }
…
```

Listing 27: Prüfung der Root ID

Als ersten Schritt, den man in Listing 27 erkenne kann, werden die beiden Root IDs in ihre drei Teile gesplittet, von denen dann die ersten verglichen werden. Sollte die zugeschickte Priorität kleiner sein, als die eigene angenommene, steht fest, dass der auswertende Switch nicht mehr der Root-Switch ist. Deshalb wird die Schriftfarbe des Switches zurück auf Blau

gesetzt und er merkt sich die zugeschickte Root ID und die Root-Pfadkosten. Da hier ein neuer Root-Switch erkannt wurde, kann der Port, der die BPDU empfangen hat, sofort zum neuen Root-Port werden. Falls es schon eine alten gab, werden zuerst alle Ports zurück auf Designated gesetzt und erst dann wird der Empfänger-Port zum neuen Root-Port. Anders als im Algorithmus beschrieben, merkt sich zusätzlich auch noch jeder Port die Root ID und Root-Pfadkosten. Diese müssen aber nicht immer die aktuellen sein. Der Port merkt sich immer nur dann etwas, wenn sich durch eine empfangen BPDU etwas am Wissen des Switches verändert hat. Das bedeutet, dass sich jeder Port seine eigenen Root-Pfadkosten merkt. Dies ist auch sinnvoll, da jeder Port einen anderen Weg zum Root-Switch hat. Nur der Switch kennt dann die echten kürzesten Root-Pfadkosten, die er dann auch ausgibt. Dasselbe gilt auch für die Root ID.
Wenn der erste Teil der Root IDs jedoch gleich ist, werden der zweite und dann der dritte verglichen. Diese Prüfungen funktionieren genauso, wie die eben beschriebene und sie wurden auch gleich implementiert. Sollte nun der dritte Teil, also die Mac-Adresse auch gleich sein, bedeutet dies, dass der auswertende Switch schon den richtigen Root-Switch kennt. Also kann sich nur noch etwas bei den Ports verändern. Wenn in der BPDU Root-Pfadkosten drinstehen, die zusammen mit den Pfadkosten der letzten Wire kleiner sind, als die bisherigen, wird der Port, der diese BPDU empfangen hat zum neuen Root-Port. Dazu muss erst einmal der alte Root-Port ermittelt werden und zurück auf Designated gesetzt werden. Dann kann der neue Port zum Root-Port werden. Dies erkennt man in Listing 28.

```
// Empfangene BPDU enthält schon bekannten Root-Switch
else if (Long.parseLong(part1[2]) == Long.parseLong(part2[2])) {
// Wenn empfangenen Root-Pfadkosten kleiner sind als bisherige Root_Pfadkosten
        if (bpdu2.root_pathcosts + aport.wire.path_costs < root_pathcosts_aktuell) {
                aport.root_id_zwischen = bpdu2.root_id;
                aport.root_pathcosts_zwischen = bpdu2.root_pathcosts
                                                        + aport.wire.path_costs;
                Port bport = ports[0];
                for (int i = 0; i < ports.length; i++) {
                        // Bestimme bisherigen Root-Port
                        if (ports[i].art == Portart.ROOT_PORT) {
                                bport = ports[i];
                        }
                }
                bport.art = Portart.DESIGNATED_PORT;
                bport.label_port.setForeground(Color.BLUE);
                aport.art = Portart.ROOT_PORT;
                aport.label_port.setForeground(Color.GREEN);
                root_pathcosts_aktuell = aport.root_pathcosts_zwischen;
                bridge_id_aktuell = bpdu2.bridge_id;
                port_id_aktuell = bpdu2.port_id;
        }
```

Listing 28: Kleinere Root-Pfadkosten

Sollten die Root-Pfadkosten auch noch gleich sein, werden die Priorität und dann die ID des alten Root-Ports mit dem, der die BPDU empfangen hat, verglichen. Dies kann man in Listing 29 sehen.

```
// Wenn empfangenen Root-Pfadkosten gleich groß sind wie die bisherige Root_Pfakosten
else if (bpdu2.root_pathcosts + aport.wire.path_costs == root_pathcosts_aktuell) {
	Port bport = ports[0];
	for (int i = 0; i < ports.length; i++) {
		// Bestimme bisherigen Root-Port
		if (ports[i].art == Portart.ROOT_PORT) {
			bport = ports[i];
		}
	}
	// Wenn Port, über den BPDU empfangen wurde, kleinere Priorität
	// hat als bisheriger Root-Port
	if (aport.priority < bport.priority) {
		bport.art = Portart.DESIGNATED_PORT;
		bport.label_port.setForeground(Color.BLUE);
		aport.art = Portart.ROOT_PORT;
		aport.label_port.setForeground(Color.GREEN);
		bridge_id_aktuell = bpdu2.bridge_id;
		port_id_aktuell = bpdu2.port_id;
	}
	// Wenn Port, über den BPDU empfangen wurde, gleiche Priorität
	// hat als bisheriger Root-Port
	else if (aport.priority == bport.priority) {
		// Wenn Port, über den BPDU empfangen wurde, kleinere ID
		// hat als bisheriger Root-Port
		if (aport.id < bport.id) {
			bport.art = Portart.DESIGNATED_PORT;
			bport.label_port.setForeground(Color.BLUE);
			aport.art = Portart.ROOT_PORT;
			aport.label_port.setForeground(Color.GREEN);
			bridge_id_aktuell = bpdu2.bridge_id;
			port_id_aktuell = bpdu2.port_id;
		}
	}
}
```

Listing 29: Gleiche Root-Pfadkosten

Wie in Listing 29 zu sehen, wird zuerst der alte Root-Port ermittelt. Dann wird die Priorität von ihm mit der vom Port, der die BPDU empfangen hat, verglichen. Der mit der kleineren wird der Root-Port. Sollte diese auch gleich sein, wird die ID verglichen. Spätestens hier wird es eine eindeutige Entscheidung geben. Der Port mit der kleineren ID wird Root-Port.
Jetzt fehlen nur noch die Blocking-Ports. Diese werden direkt im Anschluss ermittelt. Dazu wird erst einmal geprüft, ob der Port, der die Nachricht empfangen hat, überhaupt zum Root-Switch führt und nicht der Root-Port ist. Sollte beides zutreffen, werden die Root-Pfadkosten des sendenden mit denen des empfangenden Switches verglichen. Aber dieses Mal fließen die Pfadkosten der letzten Wire nicht in die Prüfung mit ein. Der Port mit den längeren Root-Pfadkosten wird Blocking-Port. Sollten jedoch die Root-Pfadkosten gleich sein, wird die Bridge ID zu Rate gezogen. Da auch sie aus drei Teilen besteht, werden wie bei der Root ID alle drei Teile verglichen. Immer der Port vom Switch mit den größeren Werten wird Blocking-Port. Dies kann man in Listing 30 sehen.

```
// Ermittlung von Blocking-Ports
if (root_switch == false) {
        // Wenn empfangener Root-Switch gleich angenommener ist
        if (bpdu2.root_id.equals(root_id_aktuell)) {
                // Wenn Empfänger-Port ungleich Root-Port ist
                if (aport.art != Portart.ROOT_PORT) {
                        //Wenn empfangene Root-Pfadkosten kleiner sind
                        if (root_pathcosts_aktuell > bpdu2.root_pathcosts) {
                                aport.art = Portart.BLOCKING_PORT;
                                aport.status = Portstatus.BLOCKING;
                                aport.label_port.setForeground(Color.RED);
                        }
                        // Wenn beide Switche selbe Root-Pfadkosten haben
                        else if (root_pathcosts_aktuell == bpdu2.root_pathcosts) {
                                String[] part3 = bpdu2.bridge_id.split(Pattern.quote("."));
                                // Wenn Priorität des aktuellen Switches größer ist
                                if (priority > Integer.parseInt(part3[0])) {
                                        aport.art = Portart.BLOCKING_PORT;
                                        aport.status = Portstatus.BLOCKING;
                                        aport.label_port.setForeground(Color.RED);
                                }
                                // Wenn Priorität des aktuellen Switches gleich ist
                                else if (priority == Integer.parseInt(part3[0])) {
                                        // Wenn ID des aktuellen Switches größer ist
                                        if (id > Integer.parseInt(part3[1])) {
                                                aport.art = Portart.BLOCKING_PORT;
                                                aport.status = Portstatus.BLOCKING;
                                                aport.label_port.setForeground(Color.RED);
                                        }
                                        // Wenn ID des aktuellen Switches gleich ist
                                        else if (id == Integer.parseInt(part3[1])) {
                                                // Wenn Mac_address des aktuellen Switches
                                                // größer ist
                                           if (Integer.parseInt(mac_address) > Integer
                                                                .parseInt(part3[2])) {
                                                aport.art = Portart.BLOCKING_PORT;
                                                aport.status = Portstatus.BLOCKING;
                                                aport.label_port.setForeground(Color.RED);
                                           }
                                        }
                                }
                        }
                }
        }
}
```

Listing 30: Bestimmung der Blocking-Ports

Nun wurde die BPDU vollständig ausgewertet und der Switch startet den Timer, um in den Learning-Status wechseln zu können. In diesem Status gibt er die Root ID und die Root-Pfadkosten im Bereich zur Darstellung des Algorithmus aus. Anschließend wechselt er nach

erneutem Ablaufen des Timers in den Forwarding-Status und schickt die BPDU verändert weiter. Da man einen BPDU-Thread nicht über mehrere Ports gleichzeitig rausschicken kann, werden Kopien dieser BPDU verschickt. Aber die Switche außer der Root-Switch natürlich erzeugen nur BPDUs, wenn sie vorher auch eine empfangen haben. Dies kann man in Listing 31 sehen.

```
if ((root_pathcosts_aktuell > bpdu2.root_pathcosts)
                        && (bpdu2.message_age <= bpdu2.max_age)) {
    BPDU bpdu1;
    for (int i = 0; i < ports.length; i++) {
        // Schicke BPDUs nur über Ports, die ein Designated-Port sind
        if (ports[i].art == Portart.DESIGNATED_PORT) {
            // Schicke BPDUs nur über Ports, die im Learning-Status oder
            // im Forwarding-Status sind
            if (ports[i].status == Portstatus.FORWARDING
                                || ports[i].status == Portstatus.LEARNING) {
                // Schicke BPDU nicht an Sender zurück
                if (ports[i].id != aport.id) {
                    Port cport = ports[i];
                    bpdu1 = create_configuration_bpdu();
                    bpdu1.message_age = bpdu2.message_age + 1;
                    String id_port = new Integer(cport.id).toString();
                    if (id_port.length() == 1) {
                        id_port = "0" + id_port;
                    }
                    bpdu1.port_id = mac_address + "." + id_port;
                    bpdu1.flag = "0";
                    cport.sendBPDU(bpdu1);
                }
            }
        }
    }
}
```

Listing 31: Weiterleiten von BPDUs

Wie in Listing 31 zu sehen, gibt es beim Weiterleiten von BPDUs ein paar Einschränkungen. Es werden nur BPDUs weitergeleitet, die von einem Switch kommen, der näher am Root-Switch ist, als der aktuelle Switch. Auch werden diese Nachrichten nur über Designated-Ports verschickt, die im Forwarding- oder im Learning-Status sind. Selbstverständlich darf die BPDU auch nicht über denselben Port geschickt werden, über den sie empfangen wurde. Über alle Ports, die diese Kriterien erfüllen, werden dann die BPDUs weitergeleitet.

Ein Switch empfängt aber nicht nur BPDUs, sondern auch Nutzdaten, die im Programm Message heißen. Genauso wie bei den BPDUs werden die Messages beim Switch in einer PriorityBlockingQueue gespeichert. Diese heißt inbox3. Auch hier werden die Inhalte nach der Empfangszeit von den Ports eingereiht. Jedes Mal, nachdem der Switch mit der Abarbeitung einer BPDU fertig ist, prüft er, ob sich eine Message in inbox3 befindet. Dies sieht man in Listing 32.

```
// Empfangen von Message
if (!inbox3.isEmpty()) {
	message1 = inbox3.peek();
	host_port = (Port) message1.empfaenger;
	// Wenn Port im Forwarding-Status ist und nicht geblockt wird
	if (host_port.status == Portstatus.FORWARDING
					&& host_port.art != Portart.BLOCKING_PORT) {
		try {
			message1 = inbox3.take();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		host_port = (Port) message1.empfaenger;
		empfange_message();
	}
	// Wenn Port geblockt ist, lösche Message
	else if (host_port.art == Portart.BLOCKING_PORT) {
		inbox3.remove();
	}
}
```

Listing 32: Prüfung, ob Message angekommen ist

Sollte eine Message in inbox3 vorhanden sein, wird, wie in Listing 32 zu erkennen, erst einmal geprüft, ob der Port, der die Message empfangen hat im Forwarding-Status und kein Blocking-Port ist. Bei erfolgreicher Prüfung wird die Message aus inbox3 genommen und dort gelöscht. Dann wird auch hier eine Methode aufgerufen, die die Message auswertet. Hier heißt sie empfange_message(). Sollte der Port, der die Message empfangen hat, ein Blocking-Port sein, wird die Message sofort aus der Liste gelöscht und auch nicht ausgewertet.

In der Methode empfange_message() wird zuerst die Zuordnung zwischen der Host-Mac-Adresse und dem Port, der die Message empfangen hat, in die HashMap sat_mac_table des Switches eingefügt. Die Mac-Adresse dient hier als Schlüssel. Die Zuordnung zwischen Host-Mac-Adresse und Empfangszeit wird in die HashMap time_table des entsprechenden Ports eingefügt. Auch hier ist die Mac-Adresse der Schlüssel.
Bevor der Switch die Message weiterschickt, löscht er alle zu alten Einträge in der SAT-Mac Table. Da in dieser HashMap die Empfangszeit nicht enthalten ist, benötigt man die HashMap time_table. Das Löschen der zu alten Einträge wird jedoch nur durchgeführt, wenn die Checkbox „Activate Aging of SAT-Mac Table entries“ gewählt wurde. Sollte dies der Fall sein, werden alle Einträge aus der HashMap time_table gesucht, die älter als 30 Sekunden sind und in der ArrayList delete_table gespeichert. Anschließend werden alle Einträge aus der HashMap sat_mac_table entfernt, die in der delete_table enthalten sind. Dies kann man in Listing 33 erkennen.

```
public void empfange_message() {
        // Aktualisiere Set-Table und Time-Table
        host_port.time_table.put(message1.sender_mac_address,
                                                        System.currentTimeMillis());
        sat_mac_table.put(message1.sender_mac_address, host_port);

        // Finde alle Einträge in der SAT-MAC-Table, die älter als 30 Sekunden sind
        if (GUI.checkbox_aging.isSelected()) {
                delete_table = new ArrayList<Entry<String, Long>>();
                for (int i = 0; i < ports.length; i++) {
                        for (Entry<String, Long> key : ports[i].time_table.entrySet()) {
                                Long empfangszeit = key.getValue();
                                // Wenn Eintrag zu alt ist
                                if (System.currentTimeMillis() > empfangszeit + 30000) {
                                        synchronized (delete_table) {
                                                delete_table.add(key);
                                        }
                                }
                        }
                }

                // Lösche zu alte Einträge
                for (int i = 0; i < delete_table.size(); i++) {
                        Entry<String, Long> zeile = delete_table.get(i);
                        host_port.time_table.remove(zeile.getKey());
                        sat_mac_table.remove(zeile.getKey());
                }
        }
        …
```

Listing 33: Löschen zu alter Einträge aus SAT-Mac Table

Nachdem nun alle zu alten Einträge aus der SAT-Mac Table gelöscht wurden, wird nun die Message weitergeleitet. Dazu wird erst einmal geprüft, ob die Message nicht schon zu alt ist. Sollte dies nicht der Fall sein, wird in der HashMap sat_mac_table nachgeschaut, ob der Ziel-Host schon bekannt ist. Wenn dies zutrifft, wird die Message über alle Designated- und Root-Ports rausgeschickt, die im Forwarding-Status sind. Auch diesmal wird die Nachricht nicht über den Port geschickt, über den sie empfangen wurde. Sollte der Ziel-Host noch nicht bekannt sein, wird ein Broadcast über alle Ports gemacht. Auch hierbei gelten dieselben Kriterien für die Ports wie eben. Da auch die Messages Threads sind, gibt es hier dasselbe Problem wie bei den BPDUs. Eine Message kann nicht gleichzeitig über mehrere Ports verschickt werden. Deshalb brauch man eine Methode die neue Messages mit demselben Inhalt erzeugt. Dies macht die Methode create_message(). Im Anschluss werden die Messages mit der Methode send_message() an die Ports übergeben. Dies kann man in Listing 34 sehen.

```
// Weiterleiten der Message
message1.message_age = message1.message_age + 1;
// Wenn Message noch nicht zu alt ist
if (message1.message_age < message1.max_age) {
	// Wenn Zielhost in Table vorhanden ist
	if (sat_mac_table.containsKey(message1.ziel_mac_address)) {
		Port bport = sat_mac_table.get(message1.ziel_mac_address);
		// Schicke Message nicht über selben Port raus, über den sie
		// empfangen wurde
		if (bport.id != host_port.id) {
			// Schicke Message nur über Ports, die Designated_Port oder
			// Root_Port sind
			if (bport.art == Portart.DESIGNATED_PORT
					|| bport.art == Portart.ROOT_PORT) {
				// Schicke Message nur über Ports, die im
				// Forwarding-Status sind
				if (bport.status == Portstatus.FORWARDING) {
					Message message2 = create_message();
					bport.sendMessage(message2);
				}
			}
		}
	}
	// Wenn Zielhost noch nicht in Table enthalten ist
	else {
		// Mache Broadcast
		for (int i = 0; i < ports.length; i++) {
		Port bport = ports[i];
		// Schicke Message nicht über selben Port raus, über den sie
		// empfangen wurde
		if (bport.id != host_port.id) {
			// Schicke Message nur über Ports, die Designated_Port
			// oder Root_Port sind
			if (bport.art == Portart.DESIGNATED_PORT
					|| bport.art == Portart.ROOT_PORT) {
				// Schicke Message nur über Ports, die im
				// Forwarding-Status sind
				if (bport.status == Portstatus.FORWARDING) {
					Message message2 = create_message();
					bport.sendMessage(message2);
				}
			}
		}
	}
}
```

Listing 34: Weiterleiten einer Message

Die Methode create_message() erzeugt einfach eine Message mit denselben Werten wie die der empfangenen Message.

7.7 Implementation der Ports

Da im vorherigen Unterkapitel ausführlich erklärt wurde, wie die Switches implementiert wurden, sind nun die Ports dran. Sie wurden nicht als Threads realisiert, da sie nicht viel zu tun haben. Sie haben nur vier Methoden und zwar receiveBPDU(), die BPDUs empfängt, sendBPDU(), die vom Switch aufgerufen wird, um eine BPDU zu verschicken, receiveMessage(), die Messages empfängt und zuletzt noch sendMessage(), die vom Switch aufgerufen wird, um Messages zu verschicken. Vereinfacht gesagt, leitet ein Port nur die BPDUs und Messages vom Switch auf die Wire und von der Wire zum Switch. Er fügt auch die Nachrichten beider Typen in die entsprechende HashMap des Switches. Zur Veranschaulichung folgt nun Listing 35.

```
@Override
public synchronized void receiveBPDU(BPDU bpdu) {
	if (disabled == false) {
		bpdu.timestamp = System.currentTimeMillis();
		while (!port_of_switch.inbox1.offer(bpdu)) {
			bpdu.timestamp = System.currentTimeMillis();
		}
	}
}

public void sendBPDU(BPDU bpdu) {
	if (disabled == false) {
		wire.sendBPDU(bpdu, this);
	}
}
```

Listing 35: Empfangen und Senden von BPDUs beim Port

In Listing 35 sieht man, dass beide Methoden nur etwas machen, wenn der Port nicht disbaled ist. Wenn ein Port eine BPDU empfängt, versucht er diese in die HashMap inbox1 einzufügen. Dies versucht er solange, bis ihm dies gelingt. Zum Verschicken einer BPDU ruft er die Methode sendBPDU der Wire auf und übergibt ihr die Nachricht und sich selbst.

7.8 Implementation der Wire

Auch die Klasse Wire hat nicht viel zu tun und hat deshalb sogar nur drei Methoden. Die einzige Aufgabe die sie hat, ist das Verschicken von BPDUs und Messages. Dazu bestimmt sie das andere Ende der Wire. Dies ist notwendig, da eine Nachricht von beiden Seiten die Wire betreten kann und zum jeweils anderen verschickt werden möchte. Auch hier folgt zur Veranschaulichung ein Listing und zwar Listing 36.

```
public void sendBPDU(BPDU bpdu, Connector sender) {
        bpdu.path_costs = path_costs;
        bpdu.empfaenger = (sender == start ? end : start);
        bpdu.start();
}

public void sendMessage(Message message, Connector sender) {
        message.path_costs = path_costs;
        message.empfaenger = (sender == start ? end : start);
        message.start();
}
```

Listing 36: Senden von BPDUs und Messages bei Wire

Wie man in Listing 36 erkennen kann, bekommt die BPDU bzw. die Message die Pfadkosten und den Port bzw. den Host, der sich am anderen Ende der Wire befindet, übergeben. Danach wird der BPDU- bzw. der Message-Thread gestartet.

7.9 Implementation der Hosts

Die Hosts müssen schon etwas mehr leisten, als die beiden vorherigen Komponenten. Sie müssen Messages erzeugen, verschicken und empfangen. Die Host wurden wieder als Threads realisiert, die ebenfalls wie die Switches beim Starten des Algorithmus gestartet werden. Auf das Empfangen von Nachrichten jeglichen Typs muss hier nicht näher eingegangen werden, da die Hosts dabei schließlich nichts machen müssen. Jedoch das Erzeugen und Verschicken einer Message kann nicht so einfach abgehandelt werden. Dazu wird wieder eine run-Methode benötigt, in der alles Entscheidende geschieht.
Zuerst müssen die Host wieder warten, bis alle gestartet wurden, damit es zu keinen Problemen kommt. Der gesamte restliche Code liegt auch hier in einer großen Schleife, die solange läuft, bis der Spanning Tree Algorithmus vom Nutzer beendet wird. Da man das Altern der Einträge in der SAT-Mac Table an- bzw. abschaltet kann, könnte der zweite Fall zu unerwünschten Einträgen in dieser Tabelle führen. Um diese falschen Einträge zumindest etwas zu reduzieren, werden die Hosts direkt nach dem Starten des Algorithmus für 60 Sekunden schlafen gelegt, um ihm Zeit zu geben den Spannbaum aufzubauen. Diese 60 Sekunden reichen natürlich nur bei kleineren Netzen aus. Wenn das Netzwerk zu groß ist, muss man dann einfach das Altern der Einträge aktivieren. Dann werden nach und nach alle falschen Einträge der SAT-Mac Table gelöscht, da nachdem der Spannbaum aufgebaut wurde die Messages keine falschen Wege mehr nehmen können. Nach dem Verstreichen der ersten 60 Sekunden erzeugt und verschickt jeder Host alle 10 Sekunden Messages an alle anderen Hosts. Dies kann man in Listing 37 sehen.

```
// Solange Simulation läuft, soll Thread aktiv bleiben
while (GUI.stop_Simulation_Button.isEnabled() == true) {
	try {
		if (erster_durchlauf == true) {
			// Wenn Pause aktiviert, soll Thread warten
			if (should_wait == true) {
				synchronized (this) {
					waits = true;
					wait();
				}
			}
			sleep(60000);
			// Wenn Pause aktiviert, soll Thread warten
			if (should_wait == true) {
				synchronized (this) {
					waits = true;
					wait();
				}
			}
			erster_durchlauf = false;
		}
		// Ab jetzt soll Thread alle 10 Sekunden Messages an alle
		// anderen Hosts verschicken
		else {
			// Wenn Pause aktiviert, soll Thread warten
			if (should_wait == true) {
				synchronized (this) {
					waits = true;
					wait();
				}
			}
			sleep(10000);
			// Wenn Pause aktiviert, soll Thread warten
			if (should_wait == true) {
				synchronized (this) {
					waits = true;
					wait();
				}
			}
		}
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
	…
```

Listing 37: Warten der Hosts

Auch die Host müssen vor und nach jedem Schlafen prüfen, ob der Algorithmus pausiert wurde. Das Verschicken der Messages wurde hier ausgespart. Auch die Methoden crrate_message() und sendMessage() sind nicht besonders schwierig. Woraus eine Message aufgebaut ist, wird später erklärt.

7.10 Implementation einer BPDU

Was eine BPDU ist, wie sie aufgebaut ist und wozu sie dient, dürft mittlerweile klar sein. Erwähnenswert ist aber noch, dass sich jeder BPDU-Thread, sobald er von der Wire gestartet wird, schlafen legt. Die Länge der Schlafenszeit hängt von den Pfadkosten der Wire ab. Danach ruft der Thread die Methode receiveBPDU() des Ports bzw. des Hosts auf. Des Weiteren ist in dieser Klasse die compareTo Methode implementiert, die die eigene Empfangszeit immer mit der einer anderen BPDU vergleicht, um die richtige Sortierung in der HashMap inbox1 zu gewährleisten.

7.11 Implementation einer Message

Genau wie die BPDUs sind auch die Messages mittels Threads realisiert. Auch sie werden durch die Wire gestartet und beinhalten dieselben Methoden zum Vergleichen und Schlafen. An inhaltlichen Fakten enthält eine Message folgende Fakten:

- message_age
- max_age
- sender_mac_adresse
- ziel_mac_adresse

Was die ersten beiden bedeuten ist klar. Die sender_mac_adresse ist die Mac-Adresse des sendenden Hosts und die ziel_mac_adresse die des Host, an den die Message geschickt wird.

7.12 Implementation des Startens des Algorithmus

Den Spanning Tree Algorithmus kann man entweder über den Menüpunkt oder den Button „Start Simulation“ starten. In beiden Fällen wird die Methode start_Simulation() des jeweiligen ActionListeners aufgerufen, die die Sichtbarkeit der Menüpunkte und Buttons regelt. In dieser Methode wird unterschieden, ob der Algorithmus bei dem aktuell geladenen Netzwerk zum ersten Mal gestartet wird oder ob dies schon häufiger geschehen ist. Sollte es das erste Mal sein, wird mit Hilfe des Unparser eine Datei erstellt, die als Backup dient. Dieses Mal braucht man keinen Zielort und auch keine Namen der Datei zu vergeben, da dies automatisch geschieht. Dieses Backup dient dazu, dass die Einstellungen, die man durchgeführt hat, beim Starten des Algorithmus nicht verloren gehen. Dies würde nämlich geschehen, da beim Starten das Configuration-File neu geparst werden muss, um ein mehrmaliges fehlerfreies Starten des Algorithmus zu gewährleisten. Aus diesem Grund wird ein Backup angelegt, das dann anstelle des ursprünglichen Configuration-Files geparst wird. Beim Laden eines Netzwerks wird dieses Backup wieder automatisch gelöscht. In beiden Fällen wird dann die Methode show_algorithmus() aufgerufen. Da das Parsen und Unparsen schon vorher beschrieben wurde, wird hier nicht nochmals darauf eingegangen.
In dieser Methode werden alle Komponenten wie die Labels für die Root ID und die Root-Pfadkosten und die Buttons zur Anzeige aller empfangenen BPDUs im Panel zur Darstellung des Algorithmus erzeugt. Ebenfalls werden hier alle Threads gestartet. Das Anlegen der Komponenten stellt keine Schwierigkeit dar und wird hier auch nicht weiter betrachtet. Für dieses Panel wurde das GridBagLayout verwendet.
Wenn man dann einen der Buttons anklickt wird die Methode button_received_bpdusActionPerformed() des ActionListeners aufgerufen, die ein Fenster erzeugt, in dem dann später alle BPDUs angezeigt werden sollen. Diese Methode ruft ihrerseits anzeige_bpdu() auf, die alle BPDUs im Panel des eben erzeugten Fensters anzeigt. Dazu legt sie für jeden Teil einer BPDU ein eigenes Label an, in das dann der entsprechende Wert geschrieben wird. Da alle BPDUs in der HashMap inbox2 der Klasse Switch gespeichert

sind, kann man ganz einfach alle Werte der BPDUs auslesen und in die Labels einfügen. Für das Fenster und das Panel wurde kein Layout-Manager verwendet.

7.13 Implementation des Pausierens des Algorithmus

Das Pausieren des Algorithmus war dahingegen schon etwas schwieriger. Ein großes Problem hat das häufige Schlafen der Threads bereitet, da man dann nicht den Befehl wait() ausführen kann. Deshalb wurde, wie vorher schon beschrieben, vor und nach jedem Schlafen eines Threads eine Abfrage eingebaut, die überprüft, ob der Menüpunkt oder der Button zum Pausieren des Algorithmus geklickt wurde. Auch sollte es möglich sein mit derselben Schaltfläche den Algorithmus anzuhalten und weiterlaufen zu lassen. Deshalb muss gezählt werden, ob der Button bzw. der Menüpunkt ein oder zweimal geklickt wurde. Bei einmaligen Klicken wird der Algorithmus pausiert und beim erneuten Klicken wird der Algorithmus weiterlaufen gelassen. Dabei wechselt auch die Beschriftung des Menüpunkts und das Bild des Buttons. Auch müssen im Pausemodus alle Buttons zur Anzeige der empfangenen BPDUs, der Menüpunkt Configurations und SAT-Mac Table aktiviert werden, damit man diese Funktionalitäten auch nutzen kann. Beim zweiten Anklicken müssen diese Schaltflächen natürlich wieder deaktiviert werden und die Threads müssen wieder angestoßen werden, damit sie weiterlaufen. Zur Veranschaulichung folgt nun Listing 38, wo man die Befehle zum Warten und Anstoßen der Threads sehen kann.

```
// Iteriere über alle Switche und lasse Threads warten
for (String key : switch_map.keySet()) {
	Switch aswitch = switch_map.get(key);
	aswitch.should_wait = true;
}
// Iteriere über alle Hosts und lasse Threads warten
for (String key : host_map.keySet()) {
	Host ahost = host_map.get(key);
	ahost.should_wait = true;
}
// Iteriere über alle Switche und stoße die Threads wieder an
for (String key : switch_map.keySet()) {
	Switch aswitch = switch_map.get(key);
	aswitch.should_wait = false;
	synchronized (aswitch) {
		aswitch.notify();
	}
}
// Iteriere über alle Hosts und stoße die Threads wieder an
for (String key : host_map.keySet()) {
	Host ahost = host_map.get(key);
	ahost.should_wait = false;
	if (ahost.waits == true) {
		synchronized (ahost) {
			ahost.notify();
			ahost.waits = false;
		}
	}
}
```

Listing 38: Pausieren und erneutes Starten der Threads

7.14 Implementation des Stoppens des Algorithmus

Zum Stoppen des Algorithmus gibt es nicht viel zu sagen. Zuerst wird wieder die Sichtbarkeit der verschiednen Schaltflächen geregelt und im Anschluss wird über alle Switches und Hosts iteriert und alle Threads mittels interrupt() beendet.

7.15 Implementation der SAT-Mac Tables

Was die SAT-Mac Table ist, wie und wann sie aufgerufen werden kann, wurde schon vorher erklärt. Zuerst wird ein Fenster erzeugt, in dem man sich mittels RadioButtons den Switch auswählen kann, zu dem man sich die SAT-Mac Table anschauen möchte. Zu der Implementation dieses Fensters gibt es nichts zu sagen, außer dass hierfür das BorderLayout verwendet wurde. Durch die Auswahl eines dieser RadioButtons öffnet sich ein weiteres Fenster, wo dann die SAT-Mac Table zu sehen ist. Diese Darstellung ist in der Methode sat_mac_table() der Klasse SAT_MAC_Table. Auch für dieses Fenster wurde das BorderLayout verwendet. Die Inhalte werden wieder mit Labels angezeigt. Die Ermittlung der Werte, die in die Labels eingefügt werden sollen, ist hier etwas schwieriger, da die HashMap sat_mac_table hierfür keine günstige Form aufweist. Als Schlüssel für die HashMap wurde ja die Mac-Adresse der Host verwendet. Jedoch benötigt man bei der Darstellung zuerst die Ports und dann alle Hosts, die man über ihn erreichen kann.
Deshalb wird über alle Ports des Switches iteriert und zu jedem ein Label mit deren IDs angelegt. Danach werden alle Hosts zu einem Port ermittelt. Dazu wird über die HashMap sat_mac_table() iteriert und alle passenden Mac-Adressen in einer neu angelegten ArrayList host gespeichert. Diese Bestimmung findet in der Methode getHostsOfPort statt. Nachdem man die Mac-Adressen der Hosts ermittelt hat, werden sie zusammengefügt und im entsprechenden Label ausgegeben. Das Bestimmen der Mac-Adressen und das Zusammenfügen dieser kann man in Listing 39 sehen.

```
if (!hosts.isEmpty()) {
        StringBuilder sb = new StringBuilder("<html>");
        for (int j = 0; j < hosts.size(); j++) {
                sb.append((j != 0 ? "<br>" : "") + hosts.get(j));
        }
        sb.append("</html>");
        gbc2.gridwidth = GridBagConstraints.REMAINDER;
        JLabel host_label = new JLabel();
        host_label.setText(sb.toString());
        panel_sat_mac_frame.add(host_label, gbc2);
}
…
private ArrayList<String> getHostsOfPort(Switch aswitch, Port port) {
        ArrayList<String> hosts = new ArrayList<String>();
        for (Entry<String, Port> entry : aswitch.sat_mac_table.entrySet()) {
                if (entry.getValue() == port) {
                        String[] parts1 = entry.getKey().split(Pattern.quote("."));
                        hosts.add(parts1[0]);
                }
        }
        return hosts;
}
```

Listing 39: Ermittlung der Host-Mac-Adressen

7.16 Implementation der Configurations

Auch über die Configurations wurde bereits ausführlich gesprochen. Wie man diese aufruft und was sie bedeuten, dürfte also bekannt sein. Alle Fenster, in denen man nur eine Auswahl von Komponenten treffen kann, braucht man hier nicht weiter zu betrachten, da es dort keine Probleme gibt.
Das Fenster, das die Attribute eines Switches anzeigt, wird in der Methode switch_attributes erzeugt. Wie man an all die Werte kommt, die angezeigt werden, wurde auch schon mehrfach beschrieben. Alle Werte, die man in diesem Fenster einstellen können soll, werden in einem Textfeld ausgegeben. Welche der Werte man einstellen kann und welche man nur in dem Configuration-File angeben kann, wurde auch schon genannt. Erst wenn man auf den OK-Button dieses Fensters klickt, werden die eingetragenen Werte übernommen. Wie im Kapitel 7.12 beschrieben, wird bei allen Einstellungen eine Backup-Datei erstellt. Dies trifft auch hier zu. Sobald man auf den OK-Button klickt, wird solch eine Datei erzeugt. Alle anderen Fenster, die die Attribute der Ports oder der Hosts anzeigen, wurden identisch implementiert.

8 Handhabung des Programms

Im Kapitel 8 wird zuerst erklärt, was man alles tun muss, um das Programm überhaupt auf seinem Rechner laufen zu lassen. Im nächsten Unterkapitel werden dann noch mal kurz alle Funktionen vorgestellt und es wird gezeigt, wie man auf sie zugreifen kann. Dieses Kapitel dient gleichzeitig als Manual für das Programm.

8.1 Einrichten des Programms

In diesem Unterkapitel wird erst einmal geklärt, was der Nutzer alles tun und beachten muss, um dieses Programm auf seinem Rechner verwenden zu können.
Auf der beiliegenden CD befindet sich unter anderem ein ZIP-Ordner, welches Programm heißt. Alle Dateien aus diesem Ordner müssen in dasselbe Verzeichnis auf dem Rechner entpackt werden. Dies wären die Jar-Datei, alle png-Bilder und das PDF-Dokument Manual. Dieser Schritt ist notwendig, damit das laufende Programm auf all diese Ressourcen zugreifen kann und richtig funktioniert.
Auch benötigt man mindestens ein Configuration-File, denn ohne solch eines kann man die eigentliche Aufgabe des Programms nicht nutzen. Wenn man keine Lust hat, selber eine zu erstellen, kann man einfach ein vorgefertigtes nehmen, das sich auf der CD befinden. Um sich mit dem Programm erst einmal vertraut zu machen, ist es auch sinnvoll, zuerst die vorgefertigten Configuration-Files zu verwenden. An das Erstellen eigener Files sollte man sich erst später wagen, da dies trotz ausführlicher Erklärung nicht so einfach ist.

8.2 Bedienung des Programms

Nun werden noch mal alle Funktionen des Programms kurz vorgestellt und gezeigt wo man sie aufrufen kann.

Laden eines Configuration-Files

Das erste, was man nach dem Starten des Programms wohl machen sollte, ist ein Configuration-File zu laden. Dazu öffnet man das Menü File und betätigt den Menüpunkt „Load Network“. Dann öffnet sich ein Fenster, in dem man durch die Ordnerstruktur auf seiner Festplatte und durch andere angeschlossene Datenträger navigieren kann, um ein Configuration-File zu finden, das man dann endgültig ins Programm laden möchte. Nachdem man sich für eins entschieden und die Auswahl mit OK bestätigt hat, erscheint auf der rechten Seite des Hauptfensters das visualisierte Netzwerk.

Positionieren der dargestellten Komponenten

Wenn man ein Configuration-File ins Programm geladen hat, in dem keine Positionsangaben für die einzelnen Komponenten angeben wurden, dürfte das dargestellte Netzwerk wohl nicht den Wünschen entsprechen. Deshalb hat man die Möglichkeit alle Switch- und Host-Komponenten mittels Drag&Drop zu verschieben und so das gesamte Netzwerk nach seinen Vorstellungen zu positionieren. Sollte der sichtbare Bereich für das Netzwerk zu klein sein, kann man natürlich auch die Komponenten über den rechten und unteren Rand hinaus ziehen. In diesem Fall tauchen dann sofort Scrollbalken auf.

Speichern eines Netzwerks

Nachdem man sich so viel Mühe mit der Positionierung gegeben hat, sollte man das Netzwerk am Besten direkt speichern. Dies kann man ebenfalls im Menü File und zwar mit Hilfe des Menüpunkts „Save Network“ tun. Auch hier öffnet sich ein Fenster, das ähnlich dem zum Laden aussieht. Nun kann man einen beliebigen Ort wählen, wo das Configuration-File erstellt werden soll. Man kann der Datei in diesem Fenster auch einen eigenen Namen geben, muss dies aber nicht tun. Nachdem man mit OK bestätigt hat, wird am angegeben Ort ein Configuration-File erstellt, das alle Einstellungen und natürlich auch die aktuellen Positionen aller Komponenten beinhaltet.

Verändern der Einstellungen einzelner Komponenten

Wie im vorherigen Abschnitt beschrieben, werden beim Speichern alle Eintellungen mitgespeichert. Diese Einstellungen kann man im Menü Run mit dem Menüpunkt Configurations vornehmen. Nun öffnet sich ein kleines Fenster, in dem man sich entscheiden muss, ob man lieber die Attribute eines Switches oder eines Hosts anschauen bzw. verändern möchte. Sollte man sich für die Switches entschieden haben, öffnet sich jetzt ein größeres Fenster, in dem alle Switches mit ihren jeweiligen Ports angezeigt werden, zwischen denen man sich nun erneut entscheiden muss. Egal für welche Komponente man sich entscheidet, es öffnet sich ein neues Fenster, in dem dann alle ihre Attribute aufgelistet sind. Einige davon kann man nur lesen, andere hingegen sogar verändern. Erst wenn man dieses Fenster mit OK bestätigt, werden die Einstellungen gemerkt und im Programm umgesetzt.
Sollte man sich im ersten Fenster jedoch für die Hosts entschieden haben, öffnet sich auch ein Fenster. In diesem hat man dann die Wahl zwischen allen Hosts des Netzwerks. Bei Auswahl eines, öffnet sich noch ein Fenster, in dem alle Attribute des Hosts angezeigt werden. Jedoch kann man von diesen keines verändern. Auch hiernach ist es sinnvoll das Netzwerk abzuspeichern.

An-/Abschalten der Alterung der Einträge in den SAT-Mac Tables

Bevor man den Spanning Tree Algorithmus startet, sollte man sich entscheiden, ob die Einträge in den SAT-Mac Tables altern sollen oder nicht. Dies kann man über die Checkbox links neber den drei Buttons machen. Standardmäßig ist diese nicht angewählt, was bedeutet, dass die Einträge nicht altern. Dies kann aber dazu führen, dass falsche Einträge in diesen Tabellen für immer bestehen bleiben. Das Altern hat aber auch einen Nachteil und zwar, dass man wahrscheinlich nie eine vollständig ausgefüllte Tabelle sehen wird, da immer irgendein Eintrag gerade vor kurzem gelöscht wurde. Jeder muss selber entscheiden, was er lieber hat.

Starten des Spanning Tree Algorithmus

Nun hat man alle Einstellungen vorgenommen und kann den Spanning Tree Algorithmus starten. Dazu gibt es zwei Möglichkeiten und zwar entweder über den Menüpunkt „Start Simulation“ unter „Run“ oder mittels des Start-Buttons. Egal welchen Weg man wählt, der Algorithmus beginnt sofort zu laufen. Im linken Bereich des Hauptfensters tauchen nun auch Informationen auf, die das aktuelle Wissen der Switches anzeigen und zwar die Root ID und die Root-Pfadkosten. Wie genau der Algorithmus arbeitet wird hier nicht erklärt. Wer dazu mehr erfahren möchte, muss das Kapitel 2 auf Seite 3 lesen.

Pausieren des Spanning Tree Algorithmus

Man kann den Algorithmus natürlich jederzeit auf Pause schalten. Dies kann man ebenfalls einmal über einen Menüpunkt und zwar diesmal den Menüpunkt „Pause Simulation“ oder über den Pause-Button erledigen. Sobald der Algorithmus pausiert, schalten sich mehrere bisher deaktivierte Schaltflächen frei. Diese sind der Menüpunkt „SAT-Mac Table“ unter „Run“ und die Buttons „Show all received BPDUs“ im linken Bereich des Hauptfensters. Jetzt kann man auch wieder Attribute unter Configurations einsehen bzw. verändern. Wenn man den Algorithmus weiterlaufen lassen möchte, muss man einfach noch mal auf eine der beiden Schaltflächen klicken.

Anzeigen aller empfangenen BPDUs

Wie im vorherigen Abschnitt beschrieben, werden die Buttons „Show all received BPDUs“ im Pausemodus aktiviert. Wenn man einen von ihnen anklickt, öffnet sich ein größeres Fenster, in dem alle von diesem Switch empfangenen BPDUs mit all ihren Bestandteilen angezeigt werden. Mit Hilfe dieser dargestellten BPDUS kann man die Arbeitsweise des Spanning Tree Algorithmus besser nachvollziehen.

Aufrufen einer SAT-Mac Table

Die andere Schaltfläche, die sich erst im Pausemodus nutzen lässt, ist der Menüpunkt „SAT-Mac Tables“. Eine SAT-Mac Table ist eine Tabelle eines Switches, in der er sich merkt, über welchen Port er welchen Host erreichen kann. Betätigt man diese Schaltfläche, öffnet sich ein Fenster, in dem alle Switches des Netzwerks aufgelistet sind. Nun muss man sich für einen davon entscheiden und dann öffnet sich ein neues Fenster, in dem endlich die SAT-Mac Table des Switches zu sehen ist. Egal ob man die Alterung nun an- oder abgeschaltet hat, wird man auch nachdem der Algorithmus fertig ist, noch einige Zeit warten müssen, bis jeder Switch gelernt hat, wo welcher Host zu finden ist.

Beenden des Spanning Tree Algorithmus

Beenden kann man den Algorithmus natürlich zu jeder Zeit. Die Statusinformationen und die Visualisierung des Netzwerks bleiben dadurch unverändert. Erst nach erneutem Starten des Algorithmus wird wieder alles zurückgesetzt. Nachdem er beendet wurde, kann man ebenfalls alle Schaltflächen, die im Pausemodus aktiviert waren, benutzen.

Aufrufen des Handbuchs

Wenn man unter Help den Menüpunkt Manual anklickt, öffnet sich ein PDF-Dokument, das das Handbuch zu diesem Programm enthält. Dort ist beschrieben welche Funktionen das Programm alle bietet und wie man sie bedient.

Beenden des Programms

Mit der Schaltfläche Exit unter dem Menü File kann man jeder Zeit das gesamte Programm beende. Alle nicht gespeicherten Einstellungen gehen dabei verloren.

9 Ausblick und Zusammenfassung

In diesem letzen Kapitel wird ein kurzer Ausblick gegeben, inwieweit man das Programm noch erweitern könnte. Zum Schluss wird dann noch mal eine kurze Zusammenfassung über diese Bachelorarbeit gegeben.

9.1 Ausblick

Rekonfiguration des Spannbaums

Wie in mehreren Kapiteln bereits angesprochen, wurden einige Aspekte des Spanning Tree Algorithmus IEEE 802.1D für diese Bachelorarbeit ausgelassen, da sie einfach zu umfangreich gewesen wären. Zum einen wäre da die Rekonfiguration des Spannbaums zu erwähnen. Damit ist gemeint, dass wenn eine Komponente oder eine Wire des Netzwerks im laufenden Betrieb ausfällt, sich der Spannbaum automatisch rekonfiguriert. Dazu müsste man erst einmal das Ausfallen einer Komponente simulieren und dann noch das Rekonfigurieren.

Zoomen innerhalb der Visualisierung des Netzwerks

Man könnte dem Programm eine weitere Schaltfläche hinzufügen, mit deren Hilfe man innerhalb des Bereichs zur Darstellung des Netzwerks rein- und raus-zoomen kann. Dies ist vor allem bei größeren Netzwerken sinnvoll.

Verschieden Geschwindigkeiten des Algorithmus-Ablaufs

Es wäre auch sinnvoll, weitere Schaltflächen hinzuzufügen, die es einem ermöglichen die Geschwindigkeit des Spanning Tree Algorithmus zu beschleunigen oder zu verlangsamen. Bei größeren Netzwerken kann es nämlich etwas Zeit in Anspruch nehmen, bis alle Nachrichten ihr Ziel erreicht haben.

Visualisierung der Arbeitsweise des Algorithmus

Man könnte die Visualisierung des Spanning Tree Algorithmus weiter ausbauen. Eine Möglichkeit dabei wäre es auch das Verschicken der BPDUs und der Messages graphisch darzustellen.

Weitere Arten des Spanning Tree Algorithmus

Man könnte auch andere Arten des Spanning Tree Algorithmus ins Programm integrieren. Vor allem wäre es sinnvoll den Rapid Spanning Tree Algorithmus und den Multiple Spanning Tree Algorithmus hinzuzufügen, damit man alle drei Arten einfach miteinander vergleichen und ihre Unterschiede untersuchen kann.

Erstellen eines Netzwerks mittels Drag&Drop

Damit man nicht für jedes Netzwerk immer ein extra Configuration-File erstellen müsste, könnte man das Programm so erweitern, damit man sich Netzwerke mittels Drag&Drop im Hauptfenster zusammenklicken könnte.

9.2 Zusammenfassung

Das Ziel dieser Bachelorarbeit war es, ein Programm in Java zu schreiben, dass Netzwerke graphisch darstellen und auf denen man dann den Spanning Tree Algorithmus IEEE 802.1D laufen lassen kann.

Im Laufe der Zeit wurde diese doch sehr grobe Aufgabenstellung immer mehr verfeinert und man hat schnell gemerkt, dass sie in einigen Gebieten zu umfangreich für eine Bachelorarbeit war. Also wurden einfach gewisse Teile weggelassen, die man dann im Rahmen einer Masterarbeit hinzufügen könnte.

Das Programm, das am Ende dabei herauskam, ist zwar nicht so umfangreich wie zu Beginn des Entstehungsprozesses gedacht war, aber es erfüllt alle gestellten Anforderungen und man kann es auf einfache Weise bedienen und nutzen.
Nun folgt eine Auflistung der wichtigsten Funktionen des Programms:

- Visualisierung von Netzwerken
- Nachträgliches Verändern der Attribute einiger Komponenten
- Speichern von geladenen Netzwerken
- Starten, Pausieren und Beenden des Spanning Tree Algorithmus
- Darstellung des aktuellen Wissens der Switches
- Farbliche Kennzeichnung der Portarten
- Darstellung der SAT-Mac Tables

Mit Hilfe dieses Programms kann man jetzt relativ einfach die Arbeitsweise des Spanning Tree Algorithmus auf einem Netzwerk verfolgen und nachvollziehen. Es könnte vielleicht anderen Studenten beim Verstehen dieses Algorithmus behilflich sein.

Anhang A

Inhalt der CD

Auf der beiliegenden CD befinden sich die folgenden Inhalte:

- Der gesamte Quellcode des Programms
- Eine ZIP-Datei mit der ausführbaren Jar-Datei sowie den allen zur Ausführung benötigten Dateien.
- Einige exemplarische Configuration-Files
- Ein Eclipse-Export des gesamten Projekts
- Die JavaDoc
- Die schriftliche Bachelorarbeit

Anhang B

Abbildungsverzeichnis

Nummer	Name	Kapitel	Seite
01	Netzwerk nach dem Laden des Configuration File 01	2.2	4
02	Port-Stati	2.2	6
03	Netzwerk nach dem Starten des Algorithmus	2.2	11
04	Netzwerk nach dem Auswerten der ersten BPDU	2.2	12
05	Netzwerk nach dem Bestimmen des ersten Blocking-Ports	2.2	14
06	Netzwerk nach Beendigung des Algorithmus	2.2	15
07	Kopfbereich im Hauptfenster	4.2	23
08	Kopfbereich nach dem Laden eines Configuration-Files	4.2	24
09	Kopfbereich nach dem Starten des Algorithmus	4.2	24
10	Kopfbereich nach dem Pausieren des Algorithmus	4.2	24
11	Bereich zur Darstellung des Netzwerks	4.3	25
12	Bereich zur Darstellung des Spanning Tree Algorithmus	4.4	26
13	Bereich zur Darstellung des Spanning Tree Algorithmus	4.4	26
14	Hauptfenster nach Beendigung des Spanning Tree Algorithmus	4.4	27
15	Entscheidung zwischen Switch und Host	5.1	28
16	Aussuchen des Switches oder des Ports	5.1	28
17	Attribute von Switch 001	5.1	29
18	Attribute von Port 01 von Switch 001	5.1	30
19	Attribut Connected mit xxx	5.1	31
20	Hostauswahl	5.1	31
21	Attribute des Hosts 008	5.1	32
22	Auswahl des Switches zur Anzeige der SAT-Mac Table	5.2	33
23	SAT-Mac Table von Switch 001	5.2	33
24	SAT-Mac Table von Switch 002	5.2	33
25	Empfangene BPDUs von Switch 001	5.3	34
26	Empfangene BPDUs von Switch 002	5.3	35
27	Visualisiertes Netzwerk nach dem Laden des Configuration-File 01	6.3	42
28	Netzwerk nach dem Laden des Configuration-File 01 ohne Positionsangaben	6.3	43
29	Überblick über alle Klassen des Programms	7.1	44
30	Das Paket „network“ mit allen Klassen	7.1	45

Anhang C

Tabellenverzeichnis

Nummer	Name	Kapitel	Seite
01	Anfängliches Wissen der Switches	2.2	7
02	Anfängliches Wissen der Ports von Switch 001	2.2	8
03	Zuordnung von Bandbreite und STP-Pfadkosten	2.2	9
04	Aufbau einer BPDU	2.2	9

Anhang D

Listingverzeichnis

Nummer	Name	Kapitel	Seite
01	Kopfzeile eines Configuration-Files	6.2	36
02	Wurzeltag	6.2	37
03	Tags für Komponenten Switch und Host	6.2	37
04	Tags für Ports	6.2	38
05	Einfache Attribute	6.2	39
06	Positionsangaben	6.2	40
07	Fertige Configuration-File 01	6.2	41
08	Ports auf disabled gesetzt	6.2	42
09	Parsen eines Configuration-Files	7.2	47
10	Einlesen der Switch-Attribute 01	7.2	48
11	Einlesen der Switch-Attribute 02	7.2	49
12	Einlesen der Port-Attribute	7.2	50
13	Einlesen der Wire-Attribute	7.2	51
14	Bestimmen der Endkomponenten 01	7.2	52
15	Bestimmen der Endkomponenten 02	7.2	53
16	Unparsen der Switches	7.3	54
17	Unparsen der Ports	7.3	55
18	Abschließen des Unparsers	7.3	56
19	Verschieben eines Switch-Labels	7.4	57
20	Probleme durch das Verschieben	7.4	58
21	Bestimmung der Positionen der Ports	7.4	59
22	Visualisieren der Wires	7.4	60
23	Übergang eines Switches vom Listening- in den Learning-Status	7.6	62
24	Erzeugen und Verschicken der ersten BPDUs	7.6	63
25	Erzeugen einer BPDU	7.6	64
26	Prüfung, ob BPDU angekommen ist	7.6	64
27	Prüfung der Root ID	7.6	65
28	Kleinere Root-Pfadkosten	7.6	66
29	Gleiche Root-Pfadkosten	7.6	67
30	Bestimmung der Blocking-Ports	7.6	68
31	Weiterleiten von BPDUs	7.6	69
32	Prüfung, ob Message angekommen ist	7.6	70
33	Löschen zu alter Einträge aus SAT-Mac Table	7.6	71
34	Weiterleiten einer Message	7.6	72
35	Empfangen und Senden von BPDUs beim Port	7.7	73
36	Senden von BPDUs und Messages bei Wire	7.8	74
37	Warten der Hosts	7.9	75
38	Pausieren und erneutes Starten der Threads	7.13	77
39	Ermittlung der Host-Mac-Adressen	7.15	78

Anhang E

Literaturverzeichnis

[1] Eck, Klaus: Das Spanning Tree-Protokoll (STP)
http://www.it-bildungsnetz.de/fileadmin/media/1x1_Networking/PDFFiles/KI10.pdf
Version: 10.08.2009 13:14 Uhr

[2] Raschke, Patrick: Aufbau einer experimentellen Spanning Tree Umgebung
https://wiki.netlab.inf.hochschule-bonn-rhein-sieg.de/index.php/Praxissemesterprojekt_Spanning_Tree
Version: 11.08.2009 13:23 Uhr

[3] Schulte, Wolfgang: Spanning Tree
http://funkschau.info/heftarchiv/pdf/2003/fs1603/fs0316055.pdf
Version: 11.08.2009 13:25 Uhr

[4] Hein, Sebastian: Spanning Tree Algorithmus
http://www.all-about-security.de/security-artikel/netzwerk-sicherheit/nac-network-access-control/artikel/2691-spanning-tree-algorithmus/
Version: 30.08.2009 13:12 Uhr

[5] Spanning Tree Protokoll
http://de.wikipedia.org/wiki/Spanning_Tree_Protocol
Version: 11.08.2009 13:53 Uhr

[6] Hein, Sebastian: Spanning Tree-Algorithmus im Detail – Teil 1
http://www.lupocom.com/artikel/artikel-grundlagen/328-spanning-tree-algorithmus-im-detail-teil-1.html
Version: 17.01.2010 12:07 Uhr

[7] Hein, Sebastian: Spanning Tree-Algorithmus im Detail – Teil 2
http://www.lupocom.com/artikel/artikel-grundlagen/329-spanning-tree-algorithmus-im-detail-teil-2.html
Version: 17.01.2010 12:10 Uhr

[8] Hein, Sebastian: Spanning Tree Algorithmus Teil 2
http://www.all-about-security.de/security-artikel/netzwerk-sicherheit/nac-network-access-control/artikel/7235-spanning-tree-algorithmus-teil-2/
Version: 17.01.2010 13:01 Uhr

[9] Schulte, Wolfgang: Pfadfinder auf Layer 2
http://wwwlehre.dhbw-stuttgart.de/~schulte/doc/LANLINE_6-2003_62.pdf
Version: 16.02.2010 14:30 Uhr

[10] Davis, David: How do I… Speed up the switch port initialization process?
http://articles.techrepublic.com.com/5100-10878_11-6073805.html
Version: 17.02.2010 13:10 Uhr

[11] Spanning Tree Timer
http://www.dista.de/netstpclc.htm
Version: 17.02.2010 13:12 Uhr

[12] Understanding Spanning-Tree Protocol Topology Changes
https://www.cisco.com/en/US/tech/tk389/tk621/technologies_tech_note09186a0080094797.shtml
Version: 17.02.2010 13:15 Uhr

[13] Spanning tree protocol
http://en.wikipedia.org/wiki/Spanning_tree_protocol
Version: 07.03.2010 14:02 Uhr

[14] Ullenboom, Christian: Java ist auch eine Insel - 15.4 Serielle Verarbeitung mit StAX
http://openbook.galileocomputing.de/javainsel8/javainsel_15_004.htm
Version: 16.03.2010 16:07 Uhr