UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Institut für Informatik

Erweiterung des Spanning Tree Simulators

Extension of the Spanning Tree Simulator

Master-Arbeit

zur Erlangung des Grades eines
Master of Sience
im Studiengang Informatik

vorgelegt von

Andreas Sebastian Janke
andrjank@uni-koblenz.de

Betreuer:
Prof. Dr. Steigner und Dipl.-Inf. Frank Bohdanowicz,
Institut für Informatik, AG Rechnernetze, Fachbereich 4: Informatik

Koblenz, im Juni 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

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

Ort, Datum Unterschrift

Zusammenfassung

Im Rahmen dieser Masterarbeit wird der Spanning Tree Simulator um diverse Funktionalitäten erweitert. Dieser wurde in der Bachelorarbeit [Jan10b] von Andreas Sebastian Janke im Jahr 2010 entwickelt. Mithilfe eines Configuration-Files kann ein beliebiges Netzwerk ins Programm geladen und grafisch dargestellt werden. Solch ein Cofiguration-File muss in XML geschrieben werden und repräsentiert ein Netzwerk bestehend aus Switches und Hosts. Anschließend kann auf diesem Netzwerk der Spanning Tree Algorithmus IEEE 802.1D laufen gelassen werden. Anders als in der Bachelorarbeit sind jetzt nur noch die Switches als Threads realisiert, um weniger Systemressourcen zu benötigen. Nach Abschluss des Algorithmus entsteht ein Spannbaum, der die Schleifen im vorliegenden Netzwerk aufbricht. Dies ist notwendig, da diese beim Verschicken von Nachrichten zu einer stetig steigenden Auslastung des Netzes führen können.

Als Erstes werden im Rahmen dieser Masterarbeit die drei folgenden Spanning Tree Algorithmen vorgestellt:

1. Spanning Tree Algorithmus IEEE 802.1D (STA)
2. Rapid Spanning Tree Algorithmus (RSTA)
3. Multiple Spanning Tree Algorithmus (MSTA)

Nach der Veranschaulichung ihrer Funktionsweise, werden sie miteinander verglichen, um ihre Unterschiede und Gemeinsamkeiten hervorzuheben.

Anschließend folgt der praktische Teil, in dem der Spanning Tree Simulator um einige Funktionen erweitert wird. Die erste Erweiterung ist die Rekonfiguration des Spannbaums nach dem Ausfall einer Verbindungsleitung zwischen zwei Switches. Eine weitere Ergänzung ist die grafische Visualisierung des Wegs, den eine Host-Nachricht nimmt. Dazu ist es sinnvoll, dem Nutzer das manuelle Verschicken von Nachrichten zu ermöglichen.
Neben dem ursprünglichen Spanning Tree Algorithmus IEEE 802.1D gibt es noch eine ganze Reihe weiterer Spanning Tree Algorithmen. Einer davon ist der Rapid Spannig Tree Algorithmus, der im Rahmen dieser Arbeit implementiert wird. Dann kann der Nutzer nacheinander beide Algorithmen auf demselben Netzwerk laufen lassen und die Unterschiede analysieren. Um die Vorteile eines Spanning Tree Algorithmus zu veranschaulichen, können Nachrichten in Netzwerken ohne Spannbaum versendet werden. Dadurch erhält der Nutzer die einmalige Gelegenheit, die Notwendigkeit des Spanning Tree Algorithmus selbst herauszufinden.
Um die Nutzbarkeit des Simulators zu erhöhen, können Netzwerke direkt im Simulator erstellt werden, anstatt sie langwierig durch ein Configuration-File zu beschreiben.

Abstract

In this master thesis some new helpful features will be added to the Spanning Tree Simulator. This simulator was created by Andreas Sebastian Janke in his bachelor thesis [Jan10b] in 2010. It is possible to visualize networks which are defined in a configuration file. Each of them is a XML representation of a network consisting of switches and hosts. After loading such a file into the program it is possible to run the Spanning Tree Algorithm IEEE 802.1D. In contrast to the previous version only the switches are implemented as threads. When the algorithm is finished a spanning tree is built. This means, that messages cannot run into loops anymore. This is important because loops can cause a total breakdown of the communication in a network, if the running routing protocols cannot handle loops.

First of all the three following spanning tree algorithms will be explained:

1. Spanning Tree Algorithmus IEEE 802.1D
2. Rapid Spanning Tree Algorithmus
3. Multiple Spanning Tree Algorithmus

After that they will be compared with eachother to identify their differences.

In the practical part some new features will be added to the Spanning Tree Simulator. The first one will be the reorganization of the spanning tree after the failure of a wire between two switches.
The user is able to send messages from one host to another and to follow their visualized pathes. The Rapid Spanning Tree Algorithm will also be implemented and the user is able to choose which of both algorithms should be run on the network. This possibility makes it easy to analyze the differences between these two algorithms. To demonstrate the advantages of the Spanning Tree Algorithm, the possibility to deactivate all these algorithms will be added to the simulator. The user should be able to build networks in the simulator without creating a configuration file.

Danksagung

An dieser Stelle möchte ich meinen Betreuern Prof. Dr. Ebert, Prof. Dr. Steigner und Dipl.-Inf. Frank Bohdanowicz danken, dass sie mir bei meiner Masterarbeit immer hilfreich zur Seite standen und jederzeit meine Fragen beantwortet haben.
Des Weiteren möchte ich Daniel Dominik Janke danken, der mir bei diversen Problemen mit einem guten Rat zur Seite stand. Auch erklärte er sich dazu bereit meine Arbeit Korrektur zu lesen.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird zum einen beschrieben, wie diese Arbeit zustande gekommen und zum anderen, wie diese aufgebaut ist.

1.1 Motivation

Im Jahr 2010 wurde im Rahmen der Bachelorarbeit [Jan10b] von Andreas Sebastian Janke der Spaning Tree Simulator entwickelt. Der Nutzer konnte Configuration-Files in den Simulator laden, in denen jeweils ein Netzwerk bestehend aus Switches, Ports, Host und Wires in XML definiert ist. Dieses wurde dann graphisch dargestellt. Anschließend konnte er den Spanning Tree Algorithmus IEEE 802.1D auf dem angezeigten Netzwerk laufen lassen. Dieser Algorithmus läuft auf den Switches und dient dem Aufbrechen vorhandener Schleifen, indem der Nachrichtenverkehr auf bestimmten Leitungen blockiert wird, damit es zu keiner Zirkulation von Nachrichten kommt.

Schon gegen Ende der Bachelorarbeit standen eine Reihe von möglichen Erweiterungen des Simulators fest, die im Rahmen der Masterarbeit implementiert werden könnten. Diese befassten sich einmal mit der Überarbeitung der GUI und zum anderen mit der Erweiterung um weitere Spanning Tree Algorithmen. Vor allem der zuletzt genannte Punkt ist sehr interessant, da dadurch die Nutzer des Simulators in die Lage versetzt werden, die Algorithmen direkt miteinander zu vergleichen. Nach einem ersten Gespräch mit Frank Bohdanowicz wurden die zu realisierenden Aufgabenpakete ermittelt. Nach intensiver Beratung standen die folgenden Aspekte fest:

- Implementation der Rekonfiguration des Spannbaums beim Spanning Tree Algorithmus IEEE 802.1D
- Implementation des Rapid Spanning Tree Algorithmus
- Abschalten beider Algorithmen
- Verschicken von Nachrichten von einem Host zu einem anderen
- Markierung des genommenen Wegs einer Host-Nachricht
- Überarbeitung der GUI
- Erstellen eines ausführlichen Vergleichs zwischen dem ursprünglichen Spanning Tree Algorithmus, dem Rapid und dem Multiple Spanning Tree Algorithmus.

Im Laufe des Wintersemesters hat sich leider herausgestellt, dass Prof. Dr. Steigner aus gesundheitlichen Gründen seine Funktion als Betreuer nicht mehr ausüben kann. Deshalb hat sich Frank Bohdanowicz auf die Suche nach einem anderen Professor gemacht, der bereit wäre die Masterarbeit zu betreuen. Prof. Dr. Ebert war damit einverstanden, diese Rolle zu übernehmen. Als dann beiden Betreuern im Rahmen einer Präsentation der Spanning Tree Simulator aus der Bachelorarbeit und die geplanten Erweiterungen vorgestellt wurden, äußerte Prof. Dr. Ebert den Vorschlag, dass es für den Anwender einfacher wäre die Netzwerke direkt im Simulator zu erstellen, als sie in einem Configuration-File zu definieren. Deshalb wurde dieser Punkt den geplanten Erweiterungen hinzugefügt.

1.2 Aufbau der Masterarbeit

Nach dieser Einleitung werden im Kapitel 2 der Spanning Tree Algorithmus IEEE 802.1D, der Rapid Spanning Tree Algorithmus und der Multiple Spanning Tree Algorithmus ausführlich erklärt und im letzten Unterkapitel werden sie miteinander verglichen. In „Funktionsweise des Spanning Tree Simulators“ (Kapitel 3) wird der Simulator mit all seinen Funktionen kurz vorgestellt. Danach kommt die Übersicht über die Klassenstruktur des Programms (Kapitel 4), in der anhand von mehreren UML-Klassendiagrammen alle Java-Klassen erläutert werden.

Damit ist der theoretische Teil dieser Arbeit abgeschlossen. Im Kapitel 5 wird erklärt, wie eine Verbindungsleitung im dargestellten Netzwerk aktiviert bzw. deaktiviert werden kann, um dadurch eine Rekonfiguration des Spannbaums beim IEEE 802.1D (Kapitel 6) zu erreichen. Die Beschreibung des Rapid Spanning Trees ist in „Implementation des Rapid Spanning Tree Algorithmus“ (Kapitel 7) zu finden.

Nachdem beide Algorithmen ausführlich behandelt wurden, kommt das Verschicken von Host-Nachrichten in einem Netzwerk mit Spannbaum (Kapiteln 8) und ohne Spannbaum (Kapitel 9) an die Reihe.

Das letzte große Kapitel 10 beschreibt die Möglichkeit, ein Netzwerk direkt im Simulator zu erstellen, anstatt es vorher mühsam in einer XML-Datei zu definieren.

Im Kapitel 11 werden eine Reihe von überarbeiteten Funktionalitäten, die es bereits in der Bachelorarbeit [Jan10b] gab, vorgestellt.

Das Kapitel 12 umfasst das Fazit dieser Masterarbeit und einen umfangreichen Ausblick.

2 Grundlagen

In diesem Kapitel wird als Vorbereitung für den Hauptteil dieser Arbeit zunächst erklärt warum durch Aufbrechen von Schleifen aus einem Graphen ein Baum wird. Im Anschluss werden die Funktionsweisen der beiden implementierten Spanning Tree Algorithmen beschrieben. Diese sind der Spanning Tree Algorithmus IEEE 802.1D und der Rapid Spanning Tree Algorithmus. Außerdem wird hier noch auf den Multiple Spanning Tree Algorithmus eingegangen und ein Vergleich zwischen den Algorithmen erstellt.

2.1 Notwendigkeit der Spanning Tree Algorithmen

Die Aufgabe jedes Spanning Tree Algorithmus ist das Erkennen und Aufbrechen von in Netzwerken vorhandenen Schleifen. Wie dies bei den beiden implementierten Algorithmen im Detail funktioniert, kann in den Unterkapiteln 2.2 und 2.3 gelesen werden.

Zunächst wird die Problematik von Schleifen in Netzwerken beschrieben. Sie sorgen zwar einerseits für mehr Ausfallsicherheit aufgrund von redundanten Pfaden, jedoch werden Nachrichten auf ihrem Weg zum Ziel auf diese Weise vervielfacht, da die Switches auf das Mittel des Broadcasts zurückgreifen. Abgesehen von dem höheren Nachrichtenaufkommen laufen zwangsläufig einige Nachrichten im Kreis, bis sie wegen ihrem zu hohen Alter gelöscht werden. Unter Umständen kann es aufgrund des erhöhten Datenaufkommens um vollständigen Erliegen des Nachrichtenverkehrs im Netzwerk führen.

Um dies zu verhindern, brechen Spanning Tree Algorithmen vorhandene Schleifen auf. Dies bewerkstelligen sie, indem sie redundante Pfade blockieren. Der Prozess läuft nicht willkürlich, sondern nach Regeln, die in den Kapiteln 2.2 und 2.3 erläutert werden, ab. Schlussendlich werden einige der vorhandenen Leitungen nicht mehr genutzt.

Anders ausgedrückt, identifizieren die Algorithmen einen Spannbaum innerhalb des Graphen. Dieser hat die Eigenschaften, dass es keine redundanten Pfade im Graphen mehr gibt. Zur Verdeutlichung folgt nun ein kleines Beispiel, das den Zusammenhang zwischen einem Netzwerk und einem Graphen verdeutlicht. Dieses ist in Abbildung 1 zu sehen.

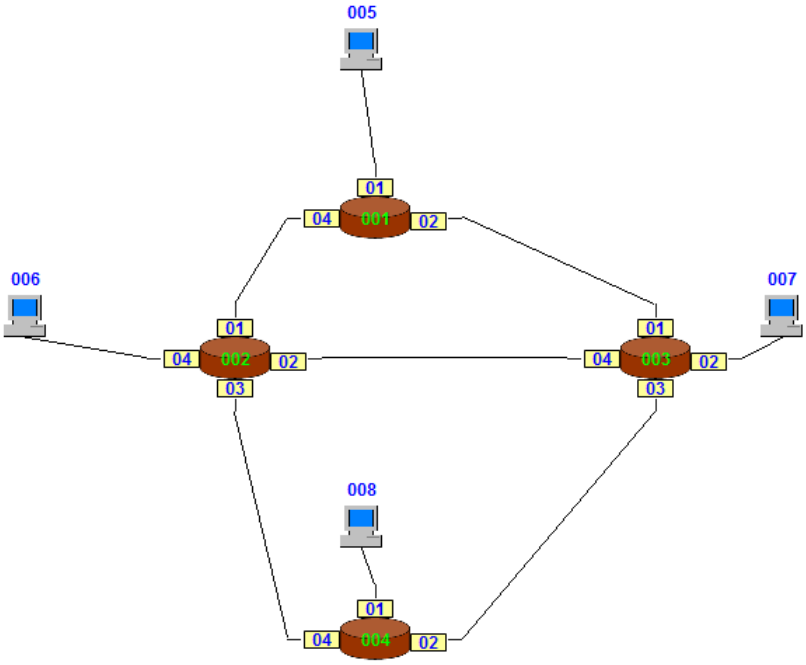


Abbildung 1: Beispielnetzwerk

Wie zu erkennen, besteht das vorliegende Netzwerk aus vier Switches, die die zentralen Komponenten darstellen. Schon hier kann die zugrundeliegende Graphstruktur erkannt werden. Um diese noch deutlicher hervorzuheben, folgt Abbildung 2.

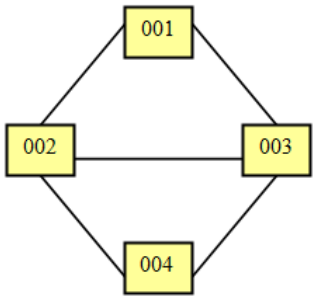


Abbildung 2: Das vorherige Netzwerk als Graph dargestellt.

Hierbei wurde sich ausschließlich auf die Switches und die Verbindungsleitungen konzentriert, damit die Grundstruktur des Netzwerks ersichtlich wird. Die Knoten repräsentieren die Switches und die Kanten die Leitungen zwischen zwei Ports. Nachdem der Spanning Tree Algorithmus zwei redundante Kanten identifiziert hat, stellen die restlichen den Spannbaum dar. Dieser ist in Abbildung 3 zu sehen.

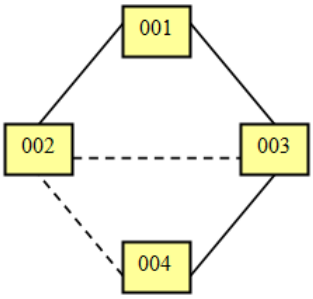


Abbildung 3: Der Spannbaum im Graphen

Die beiden gestrichelten Linien sind die als redundant identifizierten Kanten. Um den Spannbaum noch etwas deutlicher zu machen, wurden sie in der Abbildung 4 weggelassen.

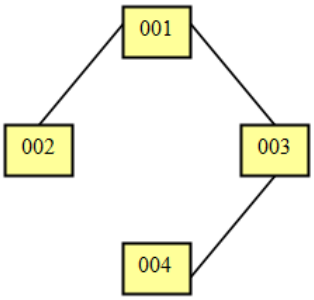


Abbildung 4: Der fertige Baum

Der Knoten 001 ist der Wurzelknoten. Alle redundanten Kanten wurden hier nicht dargestellt, sodass es von jedem Knoten aus nur genau einen Weg zu einem anderen gibt.

Diese Situation sieht als Netzwerk wie in Abbildung 5 dargestellt aus.

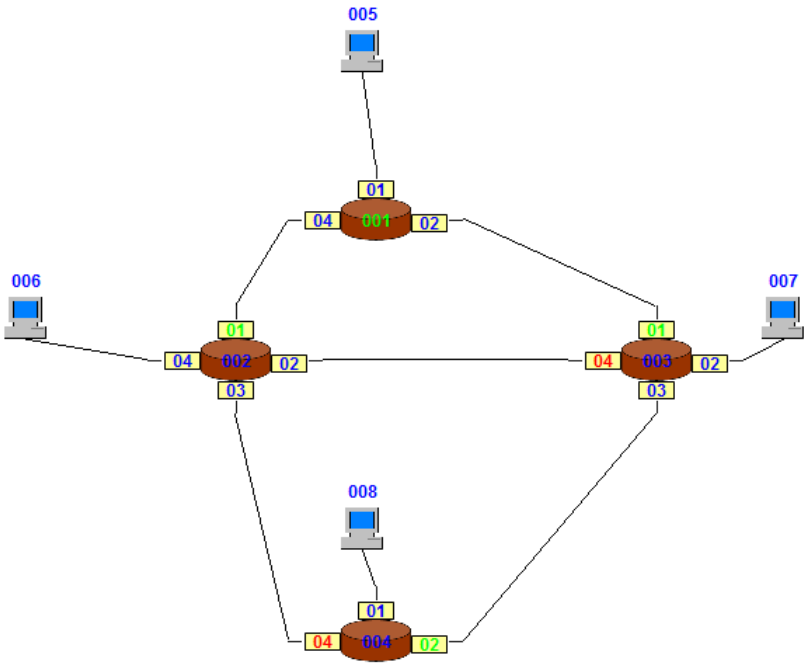


Abbildung 5: Fertiger Spannbaum

Hier sind die beiden blockierten Leitungen an der roten Einfärbung der Ports zu erkennen. Die grünen Ports zeigen die Kanten, die in Abbildung 4 zu sehen sind und den eigentlichen Spannbaum ausmachen.

Die hier beschriebene Funktionalität wird im folgenden Unterkapitel anhand des Spanning Tree Algorithmus IEEE 802.1D ausführlicher erklärt. [Jan10b]

2.2 Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D

Da in dem Unterkapitel 2.5 und dem Kapitel 6 die Kenntnis über die Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D vorausgesetzt wird, folgt hier eine kurze Beschreibung von ihm. Wer sich genauer mit diesem Algorithmus beschäftigen möchte, sollte die Bachelorarbeit [Jan10b] lesen, da er dort sehr ausführlich behandelt wird.
Alle in diesem Kapitel verwendeten Abbildungen wurden im Rahmen dieser Masterarbeit neu erstellt. Da jedoch dasselbe Netzwerk wie in der Bachelorarbeit verwendet wird, sehen einige Bilder sehr ähnlich oder unter Umständen sogar identisch aus.

In Abbildung 6 ist die Ausgangssituation für den Spanning Tree Algorithmus zu sehen.

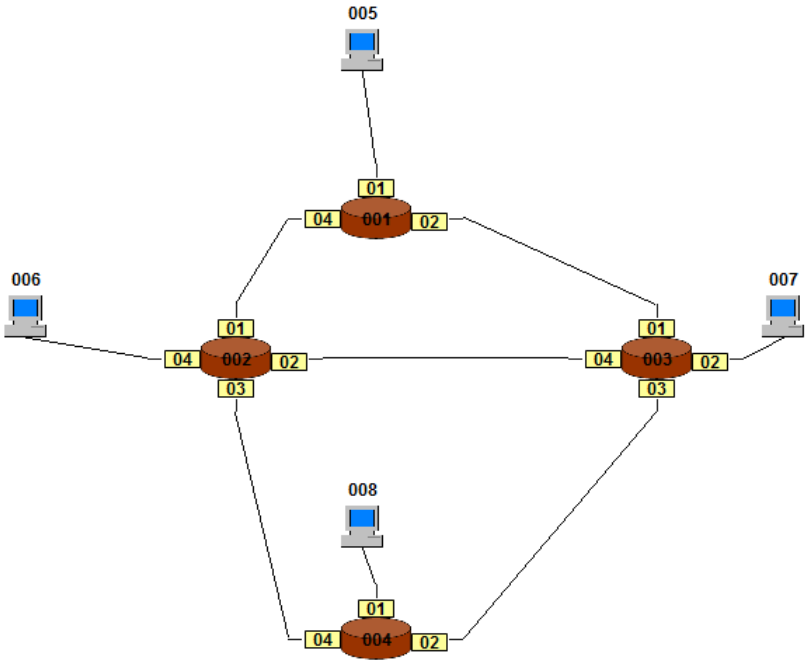


Abbildung 6: Ausgangssituation für den Spanning Tree Algorithmus IEEE 802.1D

Das in den Simulator geladene Netzwerk ist in „Configuration-File 01“[1] definiert und besteht aus vier Switches und vier Hosts. Wie zu erkennen, gibt es keine unverbundenen Komponenten und jeder Port ist an genau einer Verbindungsleitung angeschlossen. Da das Netzwerk zusammenhängend ist, kann der Spanning Tree Algorithmus IEEE 802.1D gestartet werden.

Jeder Spanning Tree Algorithmus hat zwei grundlegende Aufgaben, nämlich das Erkennen und das Aufbrechen von in Netzwerken vorhandenen Schleifen. Dazu bestimmt er aus dem vorliegenden Netz einen Spannbaum.

Die Schwierigkeit dabei ist, dass es keine übergeordnete Komponente gibt, die das gesamte Netzwerk kennt und verwaltet. Das bedeutet, dass der Algorithmus auf jedem einzelnen Switch se-

[1] Die Configuration-File 01 ist auf der beigefügten CD enthalten.

parat läuft und anfänglich auch nur über dessen Wissen verfügt. Um seine Aufgaben erledigen zu können, müssen die Switches miteinander mittels „Configuration Messages“ den sogenannten BPDUs kommunizieren und so Informationen austauschen. Wie diese Nachrichten aufgebaut sind, kann in der Bachelorarbeit [Jan10b] nachgelesen werden. Das Ziel des Algorithmus ist es, wie eben beschrieben, einen Spannbaum aufzubauen. Als Erstes wird dazu der Wurzelknoten, der hier Root-Switch heißt, ermittelt. Da direkt nach dem Start jeder Switch nur über sein eigenes Wissen verfügt, gehen demzufolge alle davon aus, dass sie selbst der Root-Switch sind. Dieser Zustand ist in Abbildung 7 dargestellt.

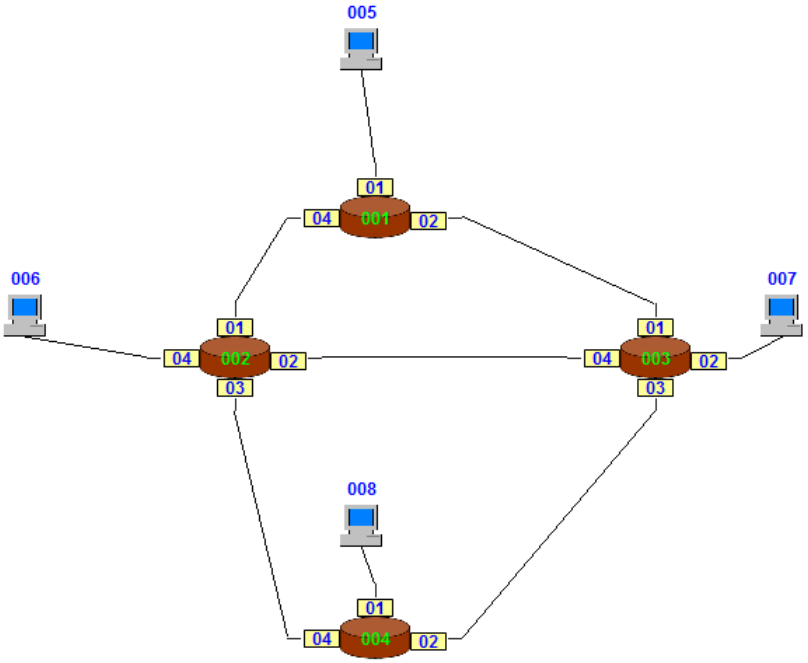


Abbildung 7: Netzwerk direkt nach dem Start des Algorithmus

Die grüne Einfärbung symbolisiert die Root-Switches und die blaue Markierung der Ports zeigt dem Nutzer, dass sie Designated-Ports sind. Diese Portart stellt sozusagen die „Default-Portart“ dar. Des Weiteren befinden sich alle Ports zu Beginn im Blocking-Status, der dafür sorgt, dass keine Nachrichten weitergeleitet werden.

Bevor die weitere Arbeit des Spanning Tree Algorithmus beschrieben wird, werden zunächst ein paar grundlegende Aspekte geklärt. Die drei folgenden Portarten spielen beim Aufbau des Spannbaums eine zentrale Rolle:

- Root-Port
- Designated-Port
- Blocking-Port

Der Root-Port stellt den kürzesten Weg zum Root-Switch dar und wird deshalb für den gesamten Nachrichtenverkehr verwendet. Der Designated-Port hat keine besondere Aufgabe und ist, wie schon vorher erwähnt, die Default-Portart. Er empfängt und leitet Nachrichten einfach weiter. Die dritte Portart dient dazu Schleifen zu blockieren, indem keinerlei Nachrichten über diesen Port verschickt werden.

Neben den Portarten gibt es noch die folgenden Portstatus, die in der Abbildung 8 veranschaulicht sind.

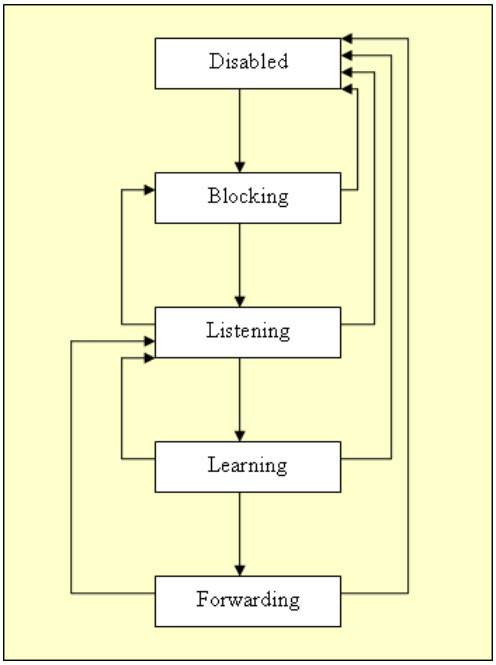


Abbildung 8: Die Portstatus des Spanning Tree Algorithmus [Sch03]

Der Disabled-Status kann vom Algorithmus selbst nicht erreicht werden und spielt deshalb hier keine Rolle. Die Aufgabe des Blocking-Status wurde bereits etwas weiter oben kurz erklärt. Im Listening-Status wertet ein Switch eine empfangene BPDU aus und im Learning-Status lernt er das empfangene Wissen. Im Forwarding-Status werden alle Nachrichten empfangen und verschickt. Nähere Informationen zu den Portarten und Portstatus sind in der Bachelorarbeit [Jan10b] zu finden.

Nachdem nun alle Switches davon ausgehen, dass sie selbst der Root-Switch sind, versetzen sie all ihre Ports schrittweise in den Forwarding-Status, um die ersten BPDUs zu verschicken. Dies ist ein etwas langwieriger Prozess, da vom Blocking-Status aus alle Status bis hin zum Forwarding-Status durchlaufen werden müssen. Bevor ein Port jedoch zum nächsthöheren wechseln kann, muss er immer einen bestimmten Zeitraum warten. Dieser ist in der Variablen „Forward Delay“ definiert und beträgt im vorliegenden Beispiel zwei Sekunden. Wenn ein Port den Forwarding-Status erreicht hat, verschickt der Switch entsprechend der Hello-Time über ihn BPDUs. Diese Variable gibt an, in welchen Zeitabständen eine BPDU vom Root-Switch erzeugt werden darf und beträgt hier sechs Sekunden.

Jede BPDU enthält eine Reihe von notwendigen Informationen, die dem Aufbau des Spannbaums dienen. Die wichtigsten davon sind:

- Root-ID
- Root-Pathcosts
- Bridge-ID
- Port-ID

Eine vollständige Liste mit ausführlichen Erklärungen befindet sich in der Bachelorarbeit [Jan10b]. In der Root-ID stehen die Priorität, die ID und die Mac-Adresse des angenommenen Root-Switches, welche momentan denen des jeweiligen Senders entsprechen. Die Root-Pathcosts enthalten die Pfadkosten zum Wurzelknoten, die jetzt noch 0 betragen. In den beiden letzten Variablen stehen Informationen über den Sender-Switch und -Port.

Wenn ein Switch solch eine BPDU empfängt, versetzt er den Empfangsport in den Listening-Status und wertet die Nachricht aus. Als Erstes wird der Root-Switch bestimmt. Dazu vergleicht der Switch die einzelnen Bestandteile der empfangenen Root-ID der Reihe nach mit denen der eigenen. Wenn beim Test herauskommt, dass der empfangene Wert kleiner als der eigene ist, bedeutet dies, dass der Sender-Switch der eigentliche Root-Switch ist. Deshalb wird die neue Root-ID mit den Root-Pathcosts übernommen. Der auswertende Switch verliert seinen Root-Switch-Status und wandelt den Empfangsport zum Root-Port um. Anschließend leitet er die BPDU über alle Designated-Ports, die sich im Forwarding-Status befinden, weiter.

Falls die eigene Root-ID kleiner ist, bleibt die auswertende Komponente Root-Switch und macht sonst nichts.
Im letzten noch verbliebenen Fall, also wenn der aktuelle Switch die empfangene Root-ID bereits kennt, muss geprüft werden, ob die Root-Pathcosts vielleicht kleiner sind als die bisherigen. Dies würde bedeuten, dass der Weg zum Root-Switch über den Empfangsport kürzer ist, als der über den bisherigen Root-Port. Wenn dem so ist, wird der Root-Port umgelegt. Sollten auch die Root-Pathcosts gleich sein, wird eine Entscheidung mit einem Vergleich der Priorität und der ID des Empfangsports mit denen des bisherigen Root-Ports erzwungen. Der Port mit den kleineren Werten wird zum Root-Port.

In Abbildung 9 ist das Netzwerk bei der Ermittlung der Root-Ports zu sehen.

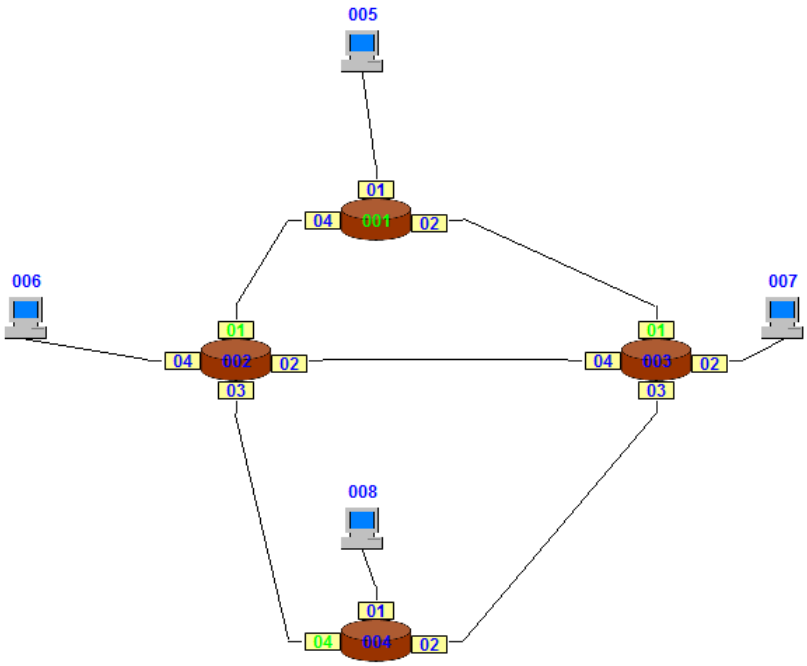


Abbildung 9: Netzwerk bei der Ermittlung der Root-Ports

Der Root-Switch 001 und die grünen Root-Ports der Switches 002 und 003 wurden bereits richtig erkannt. Jedoch Switch 004 hat noch nicht den richtigen Pfad, der über Port 02 verlaufen würde, gefunden. Der Grund weshalb er Port 02 zum Root-Port bestimmen wird, ist der, weil die beiden Pfade über dieselben Pfadkosten verfügen und er die kleinere ID hat.

Das Einzige was jetzt noch fehlt, sind die Blocking-Ports, um die redundanten Leitungen zu blockieren. Das bisher erlangte Wissen, also wer der Root-Switch und welcher Port der Root-Port ist, wird durch die folgenden Prüfungen nicht mehr verändert. Die zu prüfenden Kriterien sind:

- Root-Pathcosts
- Priorität
- ID
- Mac-Adresse

In der angegebenen Reihenfolge finden auch die Vergleiche statt. Spätestens beim Test der Mac-Adressen kann eindeutig bestimmt werden welcher der beiden Switches seinen Port blockieren muss. Anders als bei den vorherigen Prüfungen, wird hier der Switch mit den größeren Werten gewählt, da der schlechtere von beiden seinen Port blockieren soll.

Nach dem Auswerten der BPDU versetzt der Switch den Empfangsport unter Berücksichtigung des Forward-Delay-Timers in den Learning-Status und merkt sich das bisher erlangte Wissen. Anschließend wechselt der Port wieder unter Beachtung des Forward-Delay-Timers in den Forwarding-Status und leitet die eben ausgewertete BPDU über alle Designated-Ports weiter. Auf diese Weise verbreitet sich das Wissen über den Root-Switch im gesamten Netzwerk, bis schlussendlich der fertige in Abbildung 10 gezeigte Spannbaum aufgebaut ist.

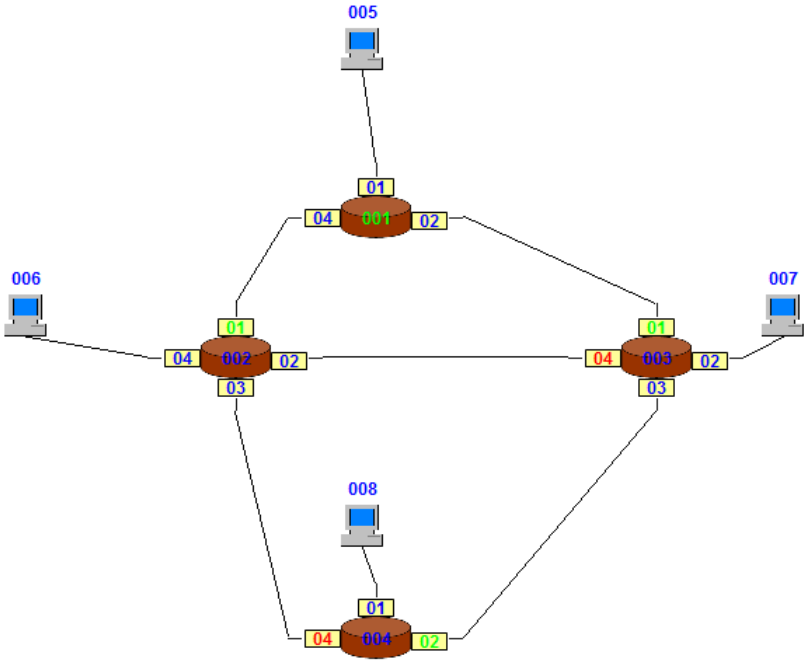


Abbildung 10: Fertiger Spannbaum

Die rot eingefärbten Ports sind hier die zwei Blocking-Ports, über die keinerlei Nachrichten verschickt und nur Configuration-Messages empfangen werden. Auf diese Weise ist sichergestellt, dass jeder Switch nur exakt einen Weg zum Root-Switch nutzt und so keine Nachrichten im Kreis laufen können.

Wem diese recht kurze Beschreibung des Spanning Tree Algorithmus nicht ausreicht, sollte die entsprechenden Kapitel in der Bachelorarbeit [Jan10b] nachlesen. [Jan10a, Jan10b]

2.3 Funktionsweise des Rapid Spanning Tree Algorithmus

In diesem Unterkapitel werden zuerst alle notwendigen Bestandteile des Rapid Spanning Tree Algorithmus ausführlich erklärt, um anschließend seine Funktionsweise grob zu veranschaulichen. Die detaillierte Beschreibung von der Funktionsweise und wie diese im Spanning Tree Simulator umgesetzt wurde, befindet sich im Kapitel 7.

Der Rapid Spanning Tree Algorithmus, auch Fast Spanning Tree Algorithmus genannt, stellt eine Weiterentwicklung des ursprünglichen Spanning Tree Algorithmus IEEE 802.1D dar. Es wurde darauf Wert gelegt, dass das neue Protokoll abwärtskompatibel ist. Damit ist gemeint, dass wenn in einem Netzwerk Switches vorhanden sind, die nur den alten Algorithmus kennen, das gesamte Netz dann diesen verwendet. Dieser neue Algorithmus hat die Bezeichnung „IEEE 802.1W“ und wurde gemeinsam mit anderen Algorithmen unter dem Standard „IEEE 802.1D-2004“ zusammengefasst. Wie die Namensteile „Rapid“ bzw. „Fast“ vermuten lassen, arbeitet er schneller als sein Vorgänger. Die größte Schwäche des ursprünglichen Protokolls war die lange Wartezeit, bis sich der Spannbaum nach dem Ausfall einer Leitung rekonfiguriert hat. Diese Dauer wurde nun mit dem weiterentwickelten Algorithmus verkürzt. [Wik11, Bad05, Cis06, eTu12, Jan10b]

2.3.1 Aufbau einer BPDU

Genau wie bei seinem Vorgänger handelt es sich beim Rapid Spanning Tree Algorithmus um ein dezentrales Protokoll. Dies bedeutet, dass es keine globale Instanz gibt, die das gesamte Netz verwaltet oder steuert. Deshalb werden auch diesmal Konfigurationsnachrichten die sogenannten BPDUs benötigt, um einen Informationsaustausch zwischen den einzelnen Switches zu ermöglichen. Da der Rapid Spanning Tree Algorithmus abwärtskompatibel sein soll, konnten die BPDUs nicht vollkommen neu konstruiert werden. Es durften also keine Felder der Nachricht weggelassen werden. Sie konnten nur etwas angepasst werden. Die Abbildung 11 zeigt den Aufbau einer BPDU. Da sich die Bezeichner der Felder nicht verändert haben und auch keine neuen hinzugekommen sind, musste die Abbildung aus der Bachelorarbeit nicht angepasst werden.

Die ersten beiden Felder „Protocol ID“ und „Version“ sind diesmal anstatt auf 1 auf 2 gesetzt, um zu verdeutlichen, dass es sich hier um BPDUs des Rapid Spanning Tree Algorithmus handelt. Der „Message Typ“ diente früher dazu festzustellen, ob die empfangene Nachricht eine normale BPDU oder eine „Topology-Change-Notification“ ist. Weiterführende Informationen zu diesem zweiten Nachrichtentyp sind im Unterkapitel 6.1 nachzulesen. Diese Unterscheidung wird im neuen Algorithmus nicht mehr getroffen. Es werden nur noch normale BPDUs verschickt und somit wäre das eben erwähnte Feld prinzipiell überflüssig, kann aber wegen der Abwärtskompatibilität nicht entfernt werden und wird deshalb konstant auf 0 gesetzt. Die Felder „Root ID“, „Root-Pathcosts“, „Bridge ID“, „Port ID“, „Message Age“, „Max. Age“, „Hello Time“ und „Forward Delay“ werden weiterhin benötigt, haben denselben Aufbau und erfüllen dieselbe Funktion wie beim ursprünglichen Spanning Tree Algorithmus.

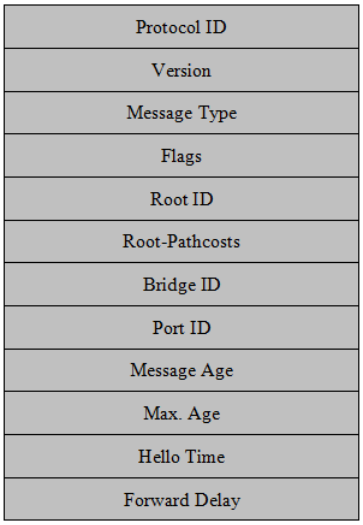


Abbildung 11: Der Aufbau einer RSTA-BPDU [Jan10b]

Auf das noch verbleibende Feld „Flags“ wird etwas ausführlicher eingegangen, da hier einige Unterschiede zu finden sind.
Beim normalen Spanning Tree Protokoll bestand dieses Feld zwar auch schon aus acht Bits, es wurden aber nur das erste und das letzte verwendet. Ihre Funktion war es den Empfänger-Switches mitzuteilen, dass eine Topologieänderung entdeckt wurde oder dass bereits eine Antwort-BPDU unterwegs ist. Weiterführende Informationen zur Rekonfiguration des Spannbaums beim Algorithmus IEEE 802.1D ist in Unterkapitel 6.1 zu finden.

Beim Rapid Spanning Tree Protokoll werden auch die noch verbleibenden sechs Bits genutzt.
Das zweite und siebte Bit bilden wie das erste und achte eine Arte Einheit. Wenn das zweite Bit auf 1 gesetzt ist, weiß der Empfänger, dass die Nachricht eine Proposal-BPDU ist und beim gesetzten siebten Bit ist es eine Agreement-BPDU. Diese beiden BPDU-Typen werden dafür benötigt, dass die Ports ohne die Forward-Delay-Timer in den Forwarding-Status wechseln können. In Unterkapitel 7.1 wird dieser Ablauf ausführlicher beschrieben. Weiterhin zu beachten wäre noch, dass niemals beide Bits gleichzeitig auf 1 gesetzt sein dürfen, da eine BPDU entweder Proposal- oder Agreement-BPDU sein kann.
Das dritte und vierte Bit gehören auch zusammen. Gemeinsam teilen sie dem Empfänger mit, welche Funktion der sendende Port ausübt. Da zwei Bits zur Verfügung stehen und jedes Bit entweder 0 oder 1 sein kann, gibt es vier verschiedene Möglichkeiten, die in der Abbildung 12 zu sehen sind.

Bit 3	Bit 4	Portart
0	0	Unknown
0	1	Alternate/Backup
1	0	Root
1	1	Designated

Abbildung 12: Die Verschlüsselung der Portart eines sendenden Ports

Die Portarten „Designated-Port“ und „Root-Port“ sind bereits bekannt, da sie im Unterkapitel 2.2 und in der Bachelorarbeit [Jan10b] ausführlich erläutert wurden. Auf die beiden neuen Portarten „Alternate-Port“ und „Backup-Port“ wird im Unterunterkapitel 2.3.3 eingegangen. [Wik11, Bad05, Cis06, eTu12, Jan10b]

2.3.2 Portstatus

Bisher wurden die BPDUs ausführlich erklärt. Als nächstes werden die Portstatus, bei denen sich im Rahmen der Entwicklung des Rapid Spanning Tree Algorithmus auch etwas verändert hat, erläutert. Diese sind in Abbildung 13 dargestellt.

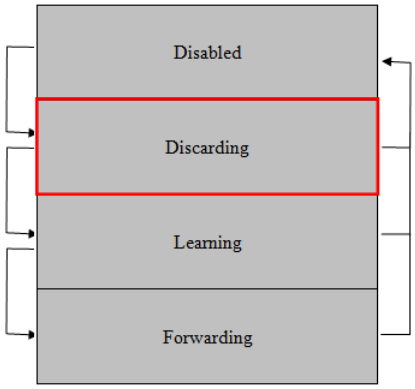


Abbildung 13: Die neuen Portstatus

Im Vergleich zu den alten Portstatus wurden der Blocking- und Listening-Status in dem neuen rot umrandeten „Discarding-Status“ vereinigt. Die Disabled-, Learning- und Forwarding-Status sind in ihrer Funktion größtenteils erhalten geblieben. Der neue Discarding-Status ist der initiale Status, den die Switches direkt nach der Initialisierung des Netzwerks einnehmen und der Einzige, in dem sie Proposal-BPDUs versenden können. Zum Auswerten, Lernen und Verschicken von BPDUs geht der sendende Port in den Learning-Status. Nachrichten von Hosts können jedoch nur im Forwarding-Status empfangen und weitergeleitet werden. Wenn ein Port vom Discarding-Status in den eben genannten Status gelangen möchte, muss er den Learning-Status durchlaufen. Beim alten Spanning Tree Algorithmus ging der Wechsel des Portstatus nur mithilfe des Forward-Delay-Timers. Dies ist beim Rapid Spanning Tree Protokoll nicht mehr notwendig, da die Ports mithilfe der Proposal- und Agreement-BPDUs direkt in den Forwarding-Status wechseln können. Der Disabled-Status kann vom Algorithmus selbst nicht verlassen oder erreicht werden. Er stellt eine vollständige Deaktivierung des betroffenen Ports dar, was nur der Nutzer selbst einstellen kann. Dabei ist es egal, welchen Status der Port vorher hatte. [Wik11, Bad05, Cis06, eTu12, Jan10b]

2.3.3 Portarten

Bei der Entwicklung des Rapid Spanning Tree Algorithmus wurden nicht nur bei den BPDUs und bei den Portstatus Änderungen vorgenommen. Es gibt auch drei vollkommen neue Portarten und zwar einmal die weiter oben erwähnten „Alternate-Ports“ und „Backup-Ports“ und dann noch die „Edge-Ports“.

Nun folgt Abbildung 14, in der ein kleines Netzwerk mit nur drei Switches dargestellt ist.

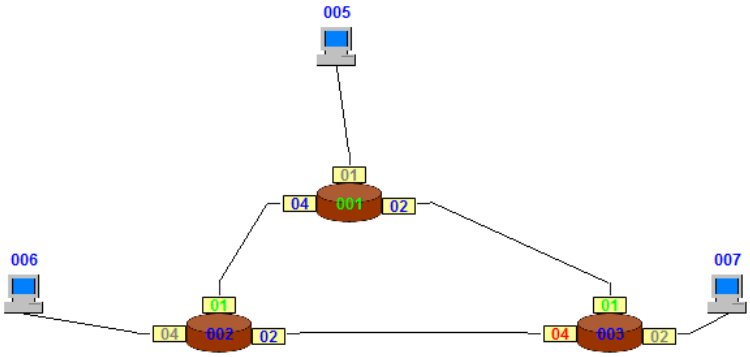


Abbildung 14: Ein Alternate-Port

Der grüne Switch mit der ID 001 ist der Root-Switch und die beiden grün eingefärbten Ports mit der ID 01 sind die Root-Ports. Diese Komponenten wurden bereits im Unterkapitel 2.2 anhand eines anderen Beispiels beschrieben.

Neu sind hingegen die grau dargestellten Ports. Diese sind die Edge-Ports, deren einzige Aufgabe darin besteht, Nutzdaten vom angeschlossenen Host zu empfangen oder an ihn zu senden. Da diese Ports mit keinem anderen Switch verbunden sind, empfangen bzw. verschicken sie auch keinerlei BPDUs, weshalb sie für den Algorithmus selbst keinerlei Rolle spielen. Sie bleiben dauerhaft in dem initialen Discarding-Status.

Auch neu hinzugekommen ist der rote Alternate-Port. Er übernimmt mit eventuell vorhandenen Backup-Ports die Aufgabe der früheren „Blocking-Ports“, was bedeutet, dass er Pfade in einem Graphen blockiert, die zu Schleifen führen würden. Wie in Abbildung 14 zu erkennen, ist Switch 003 über Port 01 direkt mit dem Root-Switch verbunden. Dies ist aber von Switch 003 aus nicht der einzige Weg zu Switch 001, da es eine zweite Verbindung zu ihm über Switch 002 gibt. Also existiert hier eine Schleife. Dies stellt Switch 003 fest, weil er über Port 01 und Port 04 BPDUs empfängt, in denen Informationen über den Root-Switch 001 enthalten sind. Anhand der bereits zuvor vorgestellten Kriterien wird zuerst der Root-Port bestimmt, der den besten Weg zum Root-Switch kennzeichnet. Nun muss der Pfad zwischen Switch 003 und Switch 002 blockiert werden. Dabei sind zwei Fragen zu klären und zwar einmal, welches Ende der Leitung blockiert werden soll und ob es ein Alternate- oder ein Backup-Port ist. Zuerst wird hier die zweite Frage beantwortet. Dazu wird geschaut, ob die zu blockierende Leitung mit demselben Switch verbunden ist wie der Root-Port. Dies trifft im vorliegenden Netzwerk nicht zu. Port 04 des Switches 003 ist mit Switch 002 und nicht mit dem Root-Switch verbunden. Also ist dieser Pfad ein alternativer und kein redundanter Weg. Nachdem bestimmt wurde, dass die Leitung eine Alternate-Leitung ist,

muss nun herausgefunden werden, welcher der beiden Ports zum Alternate-Port wird. Auf die hier beschriebene Weise wird mit allen anderen Leitungen verfahren, was dazu führen kann, dass ein Switch mehrere Alternate-Ports besitzt.

Dazu werden die folgenden Werte der beiden Switches miteinander verglichen:

- Pfadkosten zum Root-Switch
- Priorität
- ID
- Mac-Adresse

Der Switch, der bei einem der Kriterien den größeren Wert hat, muss seinen Port zum Alternate-Port deklarieren. Spätestens die Mac-Adresse sorgt für eine eindeutige Entscheidung, da jede Adresse nur genau einmal vergeben werden darf. Sobald der Port zum Alternate-Port wurde, wird er in den Discarding-Status versetzt.

Das in Abbildung 15 sichtbare Netzwerk besteht nur aus zwei Switches und der neuen Portart „Backup-Port“.

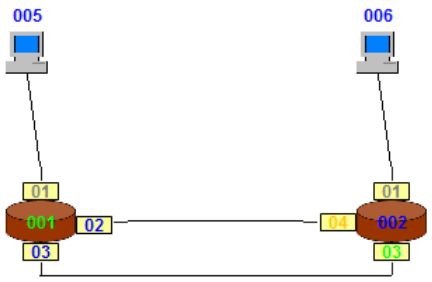


Abbildung 15: Ein Backup-Port zum Root-Port

Auch in diesem Netzwerk gibt es wieder bekannte Komponenten und zwar einen Root-Switch, einen Root-Port und zwei Edge-Ports. Das einzige Neue ist der orange eingefärbte Port 04 des Switches 002, der ein Backup-Port ist. Auch diese Ports verursachen Schleifen und müssen dementsprechend blockiert werden. Der einzige Unterschied zu einem Alternate-Port ist der, dass diese Leitung zum selben Switch führt wie der Root-Port. Also gibt es zwei redundante Pfade, von denen der ungünstigere blockiert werden muss. Auch hier muss wieder mittels der vorher schon erwähnten Kriterien die Entscheidung getroffen werden, welches der beiden Enden zum Backup-Port gemacht wird. Wie bei den Alternate-Ports kann ein Switch auch von den Backup-Ports beliebig viele haben. Sobald der Port zum Backup-Port wurde, wird er in den Discarding-Status versetzt.

Es wurden zwar jetzt alle Portarten vorgestellt, jedoch gibt es noch einen Fall, der betrachtet werden muss, und zwar, dass es mehrere Leitungen zu einem anderen Switch gibt, als den zu dem der Root-Port führt. In solch einem Fall gäbe es auch Backup-Ports zu einem Alternate-Port. In Abbildung 16 ist solch eine Situation zu sehen.

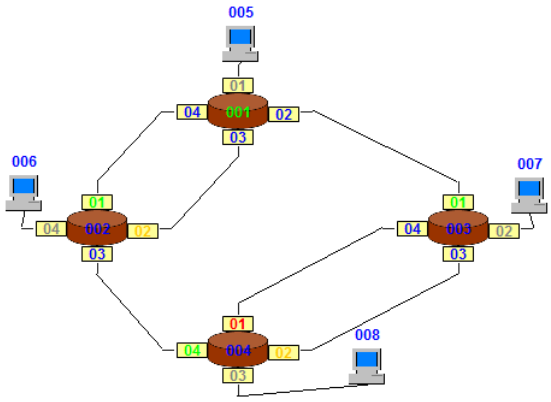


Abbildung 16: Ein Backup-Port zu einem Alternate-Port

Wie in Abbildung 16 zu erkennen, hat Switch 004 einen Alternate-Port und einen Backup-Port, die beide mit Switch 003 verbunden sind. In solch einer Situation muss herausgefunden werden, welche der beiden Leitungen zum alternativen und welche zum redundanten Pfad deklariert wird. Dazu werden die folgenden Kriterien aller in Frage kommender Ports eines Switches miteinander verglichen:

- Pfadkosten zum Root-Switch
- Priorität
- ID

Dies sind fast dieselben Kriterien, wie die zur Entscheidung welches Ende einer Leitung blockiert werden soll. Es fehlt nur die Mac-Adresse, über die Ports nicht verfügen. Durch ihre ID sind sie zumindest innerhalb eines Switches eindeutig zu bestimmen. Mittels eines lexikographischen Vergleichs dieser Kriterien wird die Alternate-Leitung bestimmt. Alle anderen werden zu Backup-Leitungen.

Als Abschluss dieses Unterunterkapitels folgt eine kurze Übersicht über die Portarten, die in Abbildung 17 dargestellt ist. [Wik11, Bad05, Cis06, eTu12, Jan10b]

Portart	Definition
Root-Port	Port mit bestem Pfad zum Wurzelknoten
Edge-Port	Port mit angeschlossenem Host
Alternate-Port	Port mit bester Sehne zu einem verbundenen Knoten, der kein direkter Eltern- oder Kindknoten ist
Backup-Port zu Root-Port	Port mit Sehne zum direkten Elternknoten
Backup-Port zu Alternate-Port	Port mit schlechterer Sehne zu einem verbundenen Knoten, der kein direkter Eltern- oder Kindknoten ist, als Alternate-Port
Designated-Port	Port mit Leitung, deren anderes Ende ein Root-, Alternate- oder Backup-Port ist

Abbildung 17: Übersicht über die Portarten

2.3.4 Überblick über die Funktionsweise des Algorithmus

Nachdem alle neuen Bestandteile des Rapid Spanning Tree Algorithmus erläutert wurden, folgt nun ein kurzer Überblick über die Funktionsweise des Protokolls.

Sobald der Algorithmus gestartet wird, gehen alle Switches davon aus, dass sie selbst der Root-Switch sind, bestimmen ihre Edge-Ports, wandeln alle anderen Ports zu Designated-Ports um und setzen sie in den Discarding-Status. Nun befinden sich alle Ports in der notwendigen Ausgangssituation, um mit dem gegenüberliegenden Port Proposal-BPDUs auszutauschen. Diese werden vom Empfänger ausgewertet und eine Kopie dieser als Agreement-BPDU zurückgeschickt. Zusätzlich setzt der Empfänger-Switch all seine anderen Ports in den Discarding-Status. Wenn ein Port die entsprechende Antwort erhält, geht er sofort in den Forwarding-Status und ist für die Weiterleitung von Host-Nachrichten bereit. Durch dieses Verfahren werden zwar mehr Nachrichten verschickt als beim alten Spanning Tree Algorithmus, aber die Ports müssen nicht immer beim Wechsel ihres Portstatus den Forward-Delay-Timer abwarten, was eine erhebliche Beschleunigung darstellt. Auf diese Weise verfahren alle Designated-Ports, die sich im Discarding-Status befinden. Das bedeutet, dass nach dem Start des Algorithmus alle Ports, abgesehen von den Edge-Ports, Proposal-BPDUs verschicken und so ihr Wissen an all ihre Nachbarn weitergeben. Nachdem die Agreement-BPDUs angekommen sind, beginnen alle Switches und nicht nur der Root-Switch entsprechend der Hello-Time normale BPDUs über all ihre Root- und Designated-Ports zu verschicken. Damit verbreitet sich das Wissen über den Root-Switch im gesamten Netzwerk und zwar ohne, dass empfangene BPDUs weitergeleitet werden müssen. Wie die diversen Portarten bestimmt werden, wurde im Unterunterkapitel 2.3.3 ausführlich erklärt.

Ein weiterer Vorteil davon, dass alle Switches BPDUs erzeugen und verschicken können, ist der, dass es jetzt eine Art Keep-Alive-Timer gibt, mit deren Hilfe ein Switch viel schneller herausbekommen kann, ob eine Verbindung unterbrochen wurde. Sollte ein Switch feststellen, dass die Leitung vom Root-Port defekt ist, tritt ein Timeout ein. Der betroffene Switch vergisst sein Wissen über den Root-Switch, also die Root-ID und die Root-Pathcosts, und schickt mittels

Broadcast an all seine Nachbarn eine BPDU, in der das erste Bit des Flag-Felds auf 1 gesetzt ist. Danach setzt er all seine Ports in die initiale Ausgangssituation zurück. Sobald ein anderer Switch solch eine BPDU empfängt, schickt er selbst über all seine Root- und Designated-Ports so eine BPDU raus und verfährt dann genauso wie sein Vorgänger. Da diese BPDUs nicht über geblockte Ports verschickt werden, muss nicht in jedem Fall der gesamte Spannbaum neu aufgebaut werden, was auch eine Beschleunigung des Algorithmus darstellt.

Die detaillierte Beschreibung der Funktionalität des Rapid Spanning Tree Algorithmus befindet sich im Kapitel 7. [Wik11, Bad05, Cis06, eTu12, Jan10b]

2.4 Funktionsweise des Multiple Spanning Tree Algorithmus

Der Multiple Spanning Tree Algorithmus stellt eine Weiterentwicklung des Rapid Spanning Tree Algorithmus dar. Er wurde unter der Bezeichnung IEEE 802.1S veröffentlicht und später mit einigen anderen Algorithmen unter IEEE 802.1Q zusammengefasst.

Der Nachteil bei den bisherigen Spanning Tree Algorithmen ist, dass aus dem gesamten Netzwerk nur ein einziger Spannbaum aufgebaut wird. Dies führt bei großen Netzwerken zu einer langwierigen Rekonfiguration. Deshalb wurde der Multiple Spanning Tree Algorithmus entwickelt. Dazu muss der Nutzer das physikalische Netz in mehrere kleinere logische Netze unterteilen, wie exemplarisch in Abbildung 18 dargestellt.

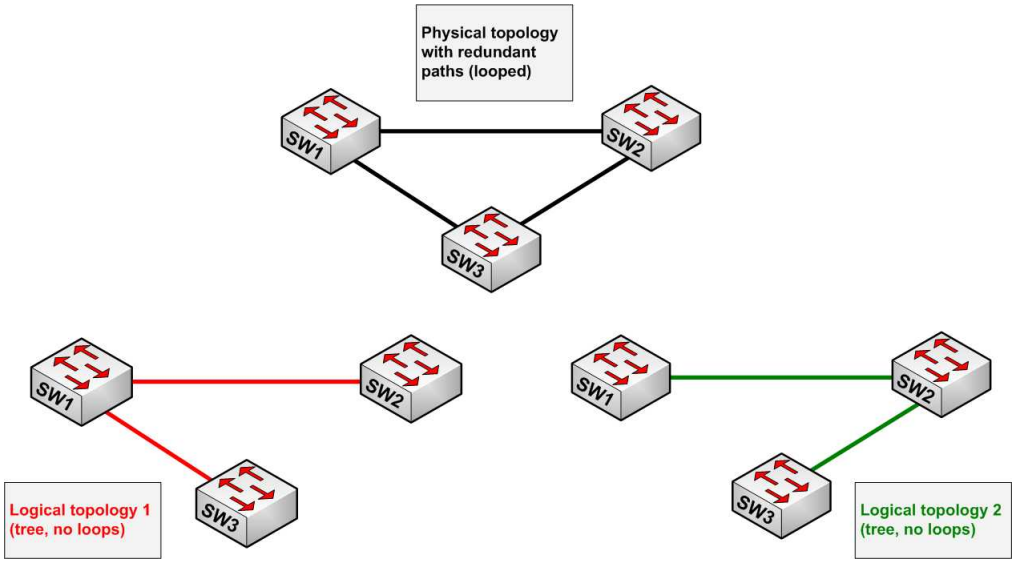


Abbildung 18: Aufteilung eines physikalischen Netzwerks in zwei logische Teilnetze
Quelle: [Lap08]

Das abgebildete physikalische Netzwerk besteht aus drei miteinander verbundenen Switches, die gemeinsam eine Schleife bilden. Nach der Aufteilung des Netzes in zwei Regionen entstehen zwei voneinander unabhängige Bäume. Wie in Abbildung 18 erkennbar, kann ein Switch gleichzeitig in mehreren Regionen sein.

Werden die beiden Bäume wieder zu einem Netzwerk zusammengeführt, sieht es wie in Abbildung 19 aus.

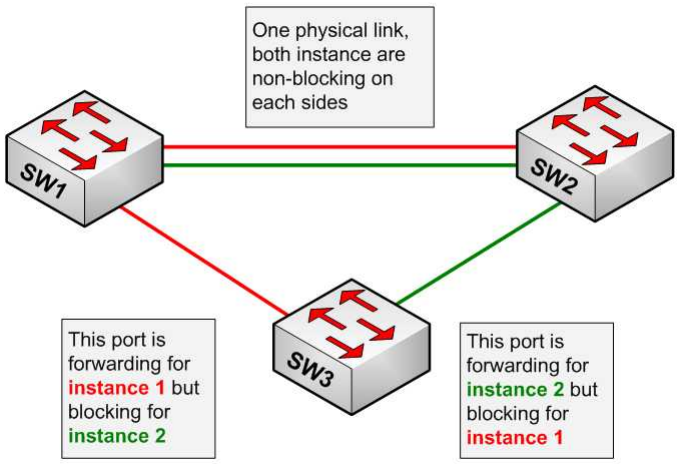


Abbildung 19: Vereinigung der beiden Teilnetze
Quelle: [Lap08]

Die Behandlung der Ports ist nun schwieriger, da sie abhängig von der angeschlossenen Region unterschiedliche Portarten und Portstatus haben können. Damit ist gemeint, dass ein und derselbe Port für die eine Region beispielsweise ein Root-Port und gleichzeitig für eine andere ein Designated-Port sein kann.

Auf jeder Region läuft eine eigene von den anderen unabhängige Instanz des Multiple Spanning Tree Algorithmus, was den eigentlichen Vorteil gegenüber seinen beiden Vorgängern darstellt. Die Rekonfiguration eines Teilnetzes ist in jedem Fall schneller, als die des gesamten Netzwerks. Wegen der strikten Trennung der einzelnen Spannbäume, ist es dem Administrator möglich das Netz so zu modifizieren, dass Nachrichten nur noch innerhalb ihrer eigenen Region laufen. Diese Aufteilung hat zur Folge, dass der Datenverkehr nicht mehr nur über einen einzigen Root-Switch läuft, was zu einem besseren Balancing führt. Dadurch ist mehr Sicherheit beim Übertragen von Nachrichten gewährleistet, weil nicht immer jeder Host alle Nachrichten empfängt.

Die eigentliche Funktionsweise des Multiple Spanning Tree Algorithmus ist ähnlich dem Rapid Spanning Tree Algorithmus und wird deshalb hier nicht näher erklärt. [Lap08, Lip]

2.5 Unterschiede und Gemeinsamkeiten der drei Spanning Tree Algorithmen

In den drei vorherigen Unterkapiteln wurden die drei Spanning Tree Algorithmen vorgestellt. Hier werden ihre Gemeinsamkeiten und Unterschiede deutlicher herauszustellen.

Schon an den ähnlichen Namen der drei Algorithmen ist zu erkennen, dass sie über gewisse Gemeinsamkeiten verfügen werden. Sie arbeiten auf den Layer 2 Switches eines Netzwerks und kommunizieren ausschließlich über BPDUs miteinander. Dies hat zur Folge, dass es kein globales Wissen gibt und sie alle dezentral arbeiten. Nur über die eben erwähnten Nachrichten ist es ihnen möglich, begrenzte Informationen über das Netz zu bekommen und das eigene Wissen zu verbreiten. Die Auswertung der empfangenen Nachrichten erfolgt vollkommen autonom. Auf diese Weise finden sie heraus welcher Switch der geeignetste Root-Switch und welcher der vorhandenen Pfade zu diesem der kürzeste ist. Dies stellt ihr gesamtes Wissen über das Netzwerk dar, das über die Spanning Tree Algorithmen gelernt wurde. Mit dieser Prozedur sind alle drei Algorithmen in der Lage Schleifen in Netzen zu erkennen. Das anschließende Aufbrechen dieser gelingt ihnen, indem sie redundante Verbindungsleitungen blockieren und nicht mehr zum Verschicken von Nachrichten verwenden. Zum Schluss etabliert sich im Netzwerk durch jeden der drei Algorithmen mindestens ein Spannbaum. Sollte eine aktive Verbindungsleitung ausfallen, wird dies registriert und mittels Rekonfiguration ein neuer Baum aufgebaut.

Wie gerade erklärt, benutzen alle drei Algorithmen das gleiche grundlegende Prinzip bei ihrer Arbeit und sogar das angestrebte Ziel, Schleifen zu erkennen und aufzubrechen, ist dasselbe. Wer sich jedoch ihre Funktionsweise näher betrachtet, dem können einige Unterschiede auffallen.

Der Spanning Tree Algorithmus IEEE 802.1D ist der erste einer ganzen Reihe von Spanning Tree Algorithmen, die größtenteils Weiterentwicklungen darstellen. Zumindest bei dem Rapid Spanning Tree und dem Multiple Spanning Tree Algorithmus ist dies der Fall. Der IEEE 802.1D legte zwar den Grundstein und war für die damaligen Netzwerke ausreichend, jedoch wurden die Netze in den darauffolgenden Jahren so groß, dass der IEEE 802.1D für die Identifizierung von Ausfällen und der anschließenden Rekonfiguration bis zu 30 Sekunden benötigte. Während dieser Aufbauprozedur bzw. der Rekonfiguration können keinerlei Host-Nachrichten also die eigentlichen Nutzdaten weitergeleitet werden, was einem gefühlten Ausfall des gesamten Netzwerks für bis zu 30 Sekunden gleichkam. Die Entwicklung eines neuen Spanning Tree Algorithmus war daher dringend notwendig. Mit dem Rapid Spanning Tree Algorithmus, der eine direkte Weiterentwicklung des IEEE 802.1D darstellt, wird der gesamte Prozess beschleunigt.

Diese Beschleunigung wird durch die folgenden Punkte ermöglicht:

- mehr BPDUs für beschleunigtes Erkennen von Leitungsausfällen
- kürzere Wege, da BPDUs nicht mehr notwendigerweise über Root-Switch laufen müssen
- weniger Timer zum Aufbau des Spannbaums
- mehr Portarten, was dazu führt, dass bestimmte Ports nicht mehr behandelt werden

Beim IEEE 802.1D kann nur der Root-Switch BPDUs entsprechend der Hello-Time erzeugen und verschicken. Diese Nachrichten werden von den anderen Switches empfangen, ausgewertet und weitergeleitet, damit das gesamte Netzwerk die Informationen bekommt. Der Rapid Spanning Tree Algorithmus funktioniert anders. Bei ihm erzeugt und verschickt jeder Switch entsprechend der Hello-Time seine eignen BPDUs, die von den Nachbar-Switches empfangen und ausgewertet werden. Ein Weiterleiten findet nicht statt. Dies sorgt für eine Beschleunigung bei dem Aufbau und der Rekonfiguration des Spannbaums.

Der IEEE 802.1D benötigt den Forward-Delay-Timer, um einen Port in den nächst höheren Portstatus zu versetzen. Dies sorgt für eine Verlangsamung des Algorithmus. Sein Nachfolger verzichtet auf diesen Timer, indem der Wechsel des Portstatus durch zusätzliche BPDUs zwischen gegenüberliegenden Ports erreicht wird. Dies erhöht zwar den Nachrichtenverkehr auf den einzelnen Leitungen, jedoch ist dies immer noch schneller als auf das Eintreten eines Timeouts warten zu müssen.

Durch den Umstand, dass beim Rapid Spanning Tree Algorithmus jeder Switch BPDUs erzeugt und verschickt, kann eine ausgefallene Leitung viel schneller entdeckt werden. Die Nachrichten werden hier als eine Art Keep-Alive-Timer verwendet. Zusätzlich verfügt der Algorithmus über mehr Portarten. Vor allem die Edge-Ports sorgen für eine geringfügige Reduzierung des Nachrichtenaufkommens, da diese beim Algorithmus einfach ignoriert und darüber keinerlei BPDUs verschickt werden.

An diesen Vorteilen ist deutlich zu erkennen, dass der Rapid Spanning Tree Algorithmus dezentraler arbeitet als der IEEE 802.1D. Beim zweitgenannten laufen alle Nachrichten ausschließlich über den Root-Switch, der die zentrale und steuernde Komponente eines Netzwerks darstellt. Beim Rapid Spanning Tree Algorithmus gibt es zwar auch einen Wurzelknoten, jedoch spielt dieser keine so zentrale Rollte mehr, da jeder Switch BPDUs erzeugen kann. Alle aufgeführten Punkte sorgen für eine deutliche Beschleunigung des Algorithmus.

Jedoch haben beide Algorithmen einen grundlegenden Nachteil, nämlich, dass bei einer steigenden Switch-Anzahl in Netzwerken diese immer mehr Zeit benötigen. Der Rapid Spanning Tree Algorithmus ist zwar schneller als der IEEE 802.1D, aber bei sehr großen Netzen ist auch er zu langsam. Deshalb wurde der Multiple Spanning Tree Algorithmus entwickelt, der eine direkte Weiterentwicklung des Rapid Spanning Tree Algorithmus ist.

Dieser neue Algorithmus funktioniert, wie weiter oben beschrieben, grundlegend ähnlich, aber der entscheidende Vorteil ist, die Aufteilung des physikalischen Netzwerks in mehrere logische Teilnetze, auf denen jeweils eine Instanz des Algorithmus arbeitet. In jedem dieser Teilnetze wird ein eigener Root-Switch bestimmt und ein unabhängiger Spannbaum aufgebaut. Da in der Regel logische Netzwerk kleiner als physikalische sind, kann der Spannbaum schneller aufgebaut und rekonfiguriert werden, was zu einer Beschleunigung führt.

In den letzten Jahren wurden noch eine ganze Reihe weiterer Spanning Tree Algorithmen entwickelt, die aber im Rahmen dieser Masterarbeit nicht betrachtet werden. [Wik11, Ava10, Bad05, Cis06, eTu12, Hei07, Hei09, Jan10a, Jan10b, Lap08, Lip, Ras08]

3 Funktionsweise des Spanning Tree Simulators

Bevor in den nächsten Kapiteln auf die Erweiterungen des in der Bachelorabeit [Jan10b] entwickelten Spanning Tree Simulators eingegangen wird, wird hier der neue Simulator mit seine Funktionen vorgestellt. Da dieser in der erwähnten Arbeit bereits ausführlich behandelt wurde, werden einige der hier verwendeten Abbildungen den alten ähneln oder mit ihnen identisch sein. Jedoch wurden sie alle komplett neu erstellt.

Nach dem Ausführen der Jar-Datei öffnet sich das Hauptfenster des Spanning Tree Simulators, das in Abbildung 20 dargestellt ist.

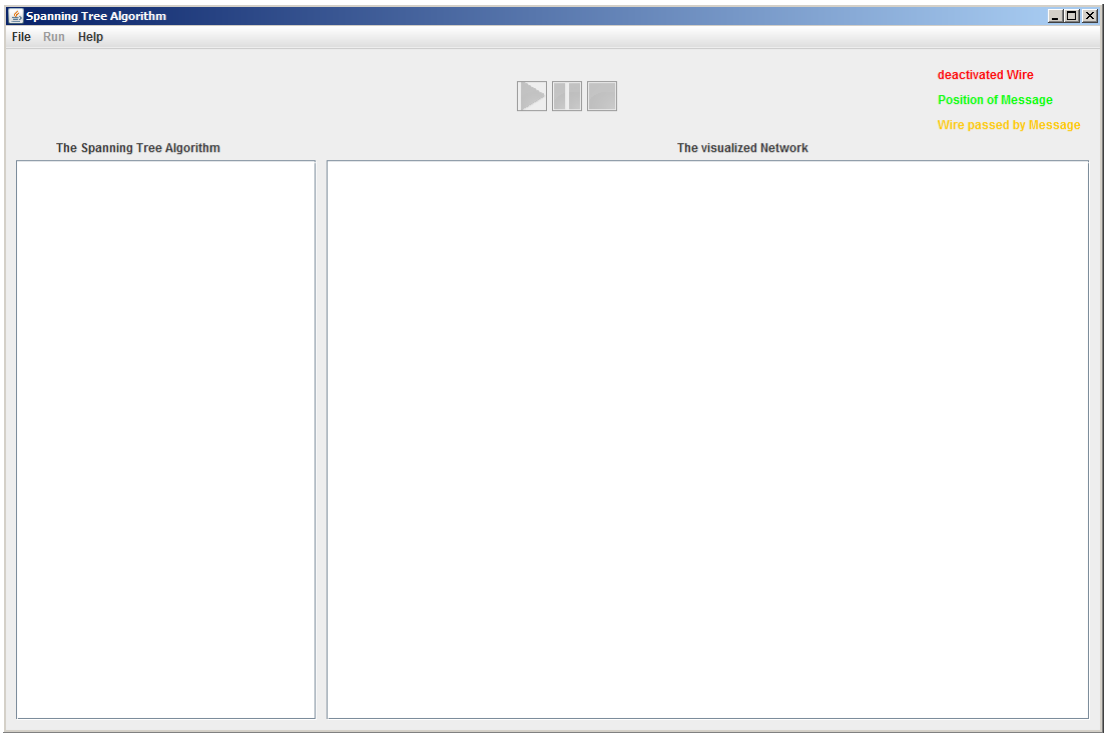


Abbildung 20: Hauptfenster direkt nach dem Start des Simulators

Wie zu erkennen, besteht das Hauptfenster aus drei Teilen und zwar aus dem oberen Steuerungsbereich, dem linken Panel zur Anzeige des Algorithmus und dem rechten Panel zur Darstellung des Netzwerks. Der Steuerungsbereich setzt sich aus dem Menü, den drei noch deaktivierten Buttons zum Starten, Pausieren und Beenden einer Simulation und ganz rechts aus der Legende zusammen, die Aufschluss über die Bedeutung der verschiedenen Leitungsfarben während einer laufenden Simulation gibt.

In der jetzigen Situation hat der Anwender zwei Möglichkeiten. Er kann entweder über den Menüpunkt „File -> Load Network“ ein Configuration-File ins Programm laden, oder er kann mittels des Kontextmenüs des rechten Panels ein neues Netz erstellen. Wie das Bauen eines neuen Netzes geht, wird in Kapitel 10 beschrieben.

Nachdem eines von beiden gemacht wurde, sieht das Hauptfenster wie in Abbildung 21 aus.

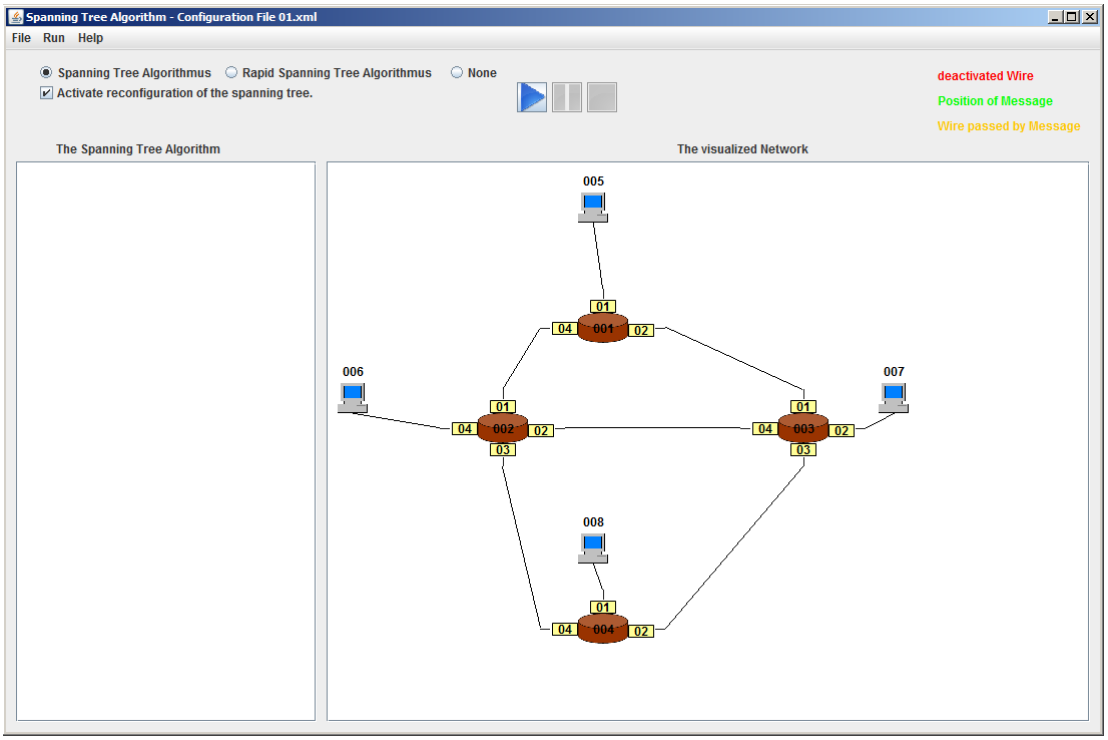


Abbildung 21: Hauptfenster nach dem Laden oder Erstellen eines Netzwerks

Die offensichtlichste Veränderung im Aussehen des Fenster ist das im rechten Panel dargestellte Netzwerk. Die dunkelbraunen rundlichen Komponenten sind die Switches, die beigen eckigen sind die Ports und die gräulichen PCs sind die Hosts. Wozu sie dienen, kann in der Bachelorarbeit [Jan10b] oder in den späteren Kapiteln nachgelesen werden. Links im Steuerungsbereich kann nun der Nutzer zwischen dem Spanning Tree Algorihtmus IEEE 802.1D, dem Rapid Spanning Tree Algorithmus und None entscheiden. Die hier abgebildete Auswahl stellt die Standarteinstellung dar. Mit den Radiobuttons kann zwischen dem Spanning Tree Algorithmus IEEE 802.1D und dem Rapid Spanning Tree Algorithmus gewechselt werden. Falls die Wahl auf None fällt, läuft nach dem Start der Simulation auf dem Netzwerk keiner der beiden Algorithmen. Nähere Informationen dazu sind im Kapitel 9 zu finden. Mithilfe der Checkbox kann in den ersten beiden Fällen die Rekonfiguration des Spannbaums an- bzw. abgeschaltet und im dritten die Alterung der SAT-MAC-Table-Einträge aktiviert bzw. deaktiviert werden.

Solange die Simulation noch nicht läuft, hat der Nutzer die Möglichkeit den Aufbau des dargestellten Netzwerks zu verändern, indem er beliebige Komponenten hinzufügt, entfernt oder verschiebt. Darüber hinaus kann er auch die Attribute der Switches, Ports und Hosts verändern. Dazu benötigt er das Configurations-Fenster, das entweder über das Menü des Hauptfensters oder über das Kontextmenü der entsprechenden Komponente aufgerufen werden kann. Falls ersteres angeklickt wird, erscheinen nacheinander zwei Fenster, in denen zuerst die gewünschte Komponente ausgewählt werden muss, bevor das endgültige Fenster mit den Attributen auftaucht. Weil dies ziemlich umständlich ist, wurde im Rahmen dieser Masterarbeit ermöglicht viele Funktionen auch über ein Kontextmenü aufzurufen. Die Abbildung 22 zeigt beispielhaft das Configurations-Fenster von Switch 002.

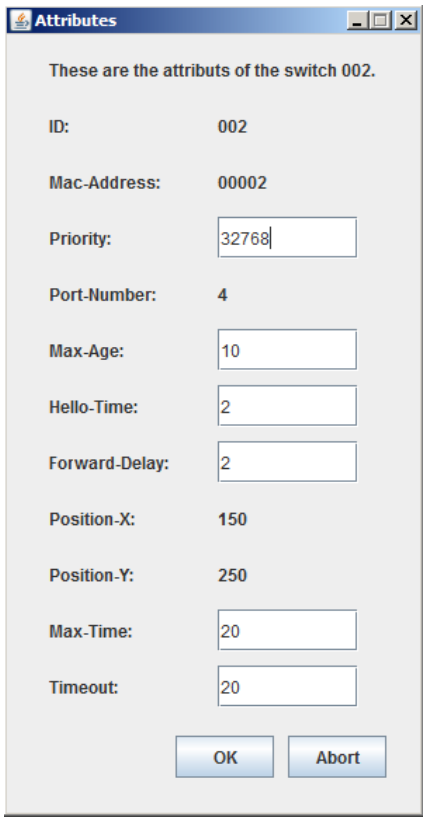


Abbildung 22: Das Configurations-Fenster von Switch 002

Das hier sichtbare Fenster wird in Unterkapitel 11.2 näher beschrieben.

Sobald die Simulation über den sichtbaren Button oder den entsprechenden Menüpunkt gestartet wurde, ist das Netzwerk nicht mehr veränderbar. Der gewählte Algorithmus beginnt nun mit seiner Aufgabe und der Nutzer kann alle Veränderungen im rechten Panel mitverfolgen. Wenn der Spannbaum fertig aufgebaut wurde, ist das Netzwerk soweit stabil und es ändert sich ohne Beeinflussung durch den Anwender nicht mehr. Das Hauptfenster mit dem fertigen Spannbaum ist in Abbildung 23 dargestellt.

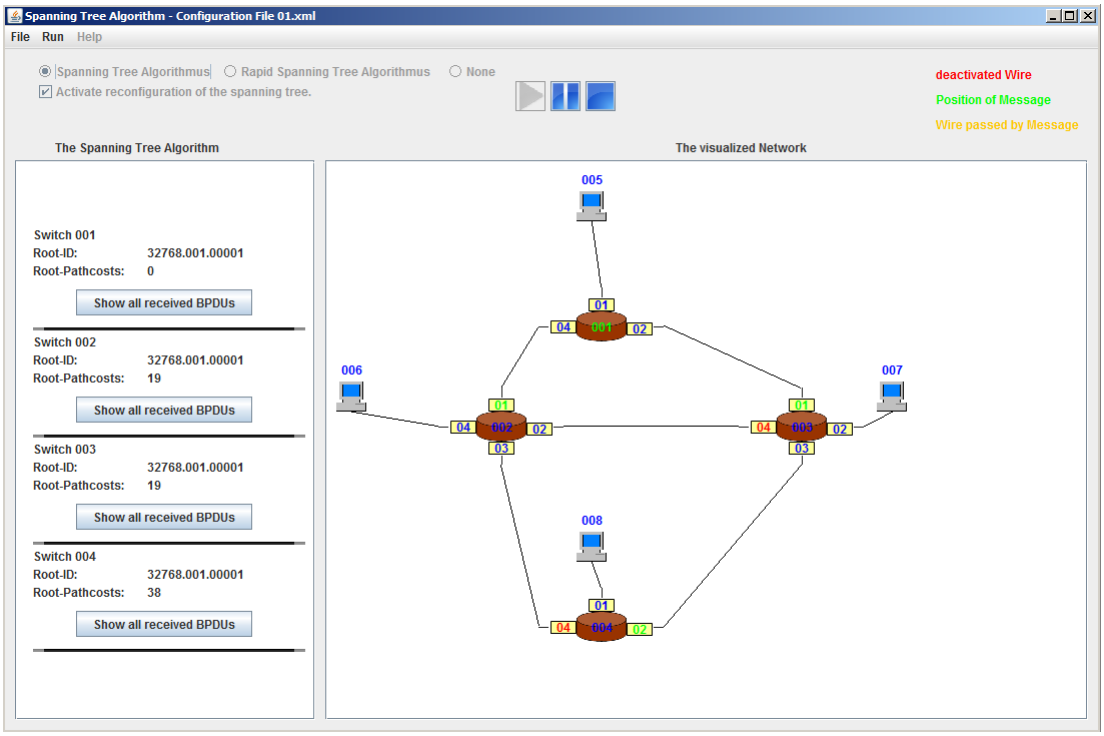
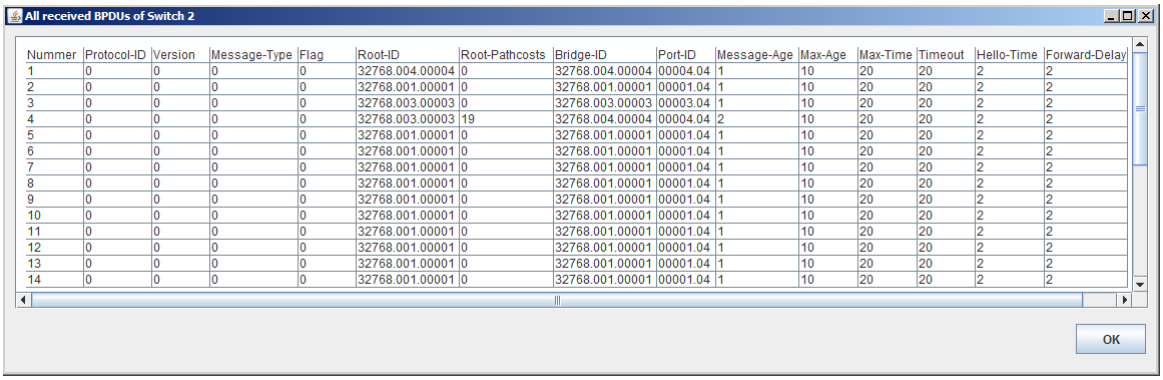


Abbildung 23: Hauptfenster mit fertigem Spannbaum

Im linken Panel erscheint direkt nach dem Start eine Übersicht über die vorhandenen Switches mit ihrer Root-ID und ihren Root-Pathcosts. Unter diesen Werten gibt es für jeden Switch einen eigenen Button, bei dessen Betätigung ein Fenster mit allen empfangenen Configuration Messages erscheint. Dieses ist in Abbildung 24 zu sehen.



Nummer	Protocol-ID	Version	Message-Type	Flag	Root-ID	Root-Pathcosts	Bridge-ID	Port-ID	Message-Age	Max-Age	Max-Time	Timeout	Hello-Time	Forward-Delay
1	0	0	0	0	32768.004.00004	0	32768.004.00004	00004.04	1	10	20	20	2	2
2	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
3	0	0	0	0	32768.003.00003	0	32768.003.00003	00003.04	1	10	20	20	2	2
4	0	0	0	0	32768.003.00003	19	32768.004.00004	00004.04	2	10	20	20	2	2
5	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
6	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
7	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
8	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
9	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
10	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
11	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
12	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
13	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
14	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2

Abbildung 24: Fenster zur Anzeige der empfangenen Configuration Messages

Nun kann der Nutzer über das Kontextmenü der Ports beliebige Leitungen ausfallen lassen und beobachten, wie der Algorithmus damit umgeht. Er hat auch die Möglichkeit über das Kontextmenü der Hosts Messages von einem Host zu einem anderen zu verschicken und ihre Bewegung durch das Netz zu verfolgen. Nähere Informationen zu diesen beiden Funktionalitäten sind in den Kapiteln 5, 6, 7, 8 und 9 ausführlich erklärt.

Durch das Verschicken von Messages lernen die Switches über welchen Port welcher Host zu erreichen ist. Dieses Wissen wird in den SAT-MAC-Tables gespeichert, die über den gleichnamigen Menüpunkt des Hauptfensters bzw. über das Kontextmenü der Switches aufgerufen werden können. In den Kapiteln 8 und 9 sind ausführliche Erklärungen hierzu zu finden. Das dazugehörige Fenster ist in Abbildung 25 zu erkennen.

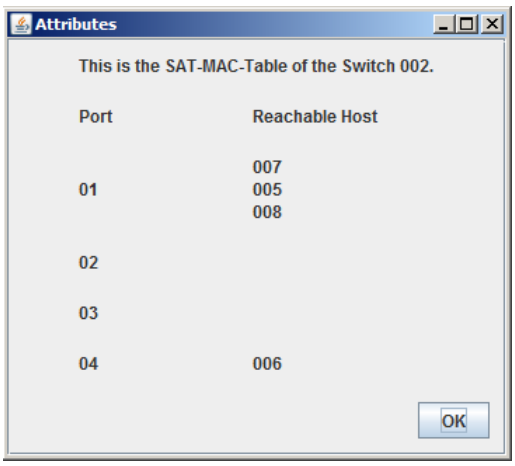


Abbildung 25: Die SAT-MAC-Table des Switches 002

Die Funktionsweise der Algorithmen oder der Weg, den eine Message nimmt, sind nicht immer auf Anhieb zu verstehen. Dafür gibt es den Pause-Modus, mit dessen Hilfe der Anwender die Simulation jederzeit anhalten kann. Das dargestellte Netzwerk kann über den Menüpunkt „Save Network“ als XML-Datei gespeichert werden. Dies ist jedoch nur möglich wenn keine Simulation läuft. Über den Menüpunkt Help ist diese Dokumentation und die zur Bachelorarbeit zu finden. [Jan10a, Jan10b]

4 Übersicht über die Klassenstruktur

In diesem Kapitel werden alle Java-Klassen vorgestellt, aus denen der Spanning Tree Simulator besteht. Zum besseren Verständnis werden die Packages „network“ und „spanningTreeAlgorithm“ getrennt voneinander betrachtet.

4.1 Package „network“

Um zu verstehen, wie die einzelnen Klassen dieses Packages zusammenhängen, folgt die Abbildung 26, in der ein UML-Klassendiagramm zu sehen ist, das einen ersten Überblick gewährt.

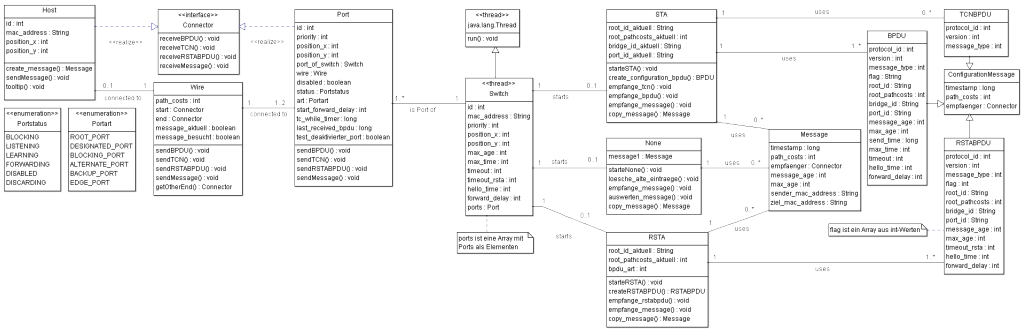


Abbildung 26: Klassendiagramm mit allen Klassen des Pakets „network“

Dieses Diagramm ist zwar gut, um dem Leser einen ersten Eindruck vom Umfang des Programms zu vermitteln, jedoch ist es zu groß, um noch gelesen werden zu können. Deshalb wurde es in zwei Teile aufgeteilt, die im Anschluss getrennt voneinander erklärt werden.

In der Abbildung 27 ist der linke Teil des Klassendiagramms 26 zu sehen. Es zeigt, wie ein Netzwerk aufgebaut wird.

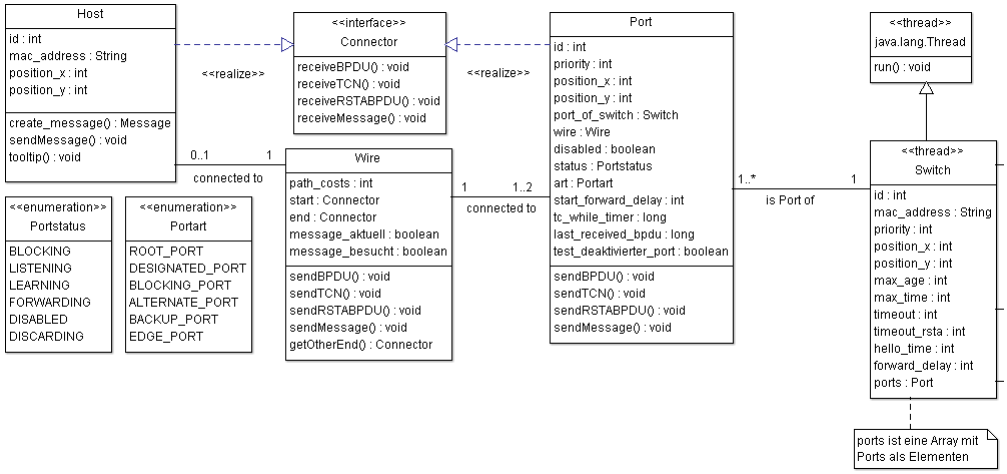


Abbildung 27: Klassendiagramm zum Aufbau eines Netzwerks

Die zentralen und somit wichtigsten Komponenten sind die Switches, die in der gleichnamigen Klasse am rechten Rand des Diagramms definiert sind. Anders als in der Bachelorarbeit [Jan10b] ist dies die einzige Klasse, die noch als Thread realisiert wird. Dort waren zusätzlich zu den Switches alle Nachrichten und Hosts auch Threads, was zu einer zu großen Systemauslastung führte. Jeder Switch besitzt eine ganze Reihe von Attributen, die er bei seiner Erzeugung übergeben bekommt. Einige davon wie beispielsweise „position_x“ werden zur Darstellung der Switch-Komponente benötigt und andere wie „priority“ werden von den Spanning Tree Algorithmen gefordert. Die angegebene Liste ist nicht vollständig, da ein Switch über zu viele Attribute verfügt, um sie alle aufzuführen.

Die zweite betrachtete Klasse symbolisiert die Ports. Jeder Switch muss mindestens einen Port haben. Nach oben ist die Anzahl theoretisch unbegrenzt, jedoch gibt es nur vier Positionen, die ein Port einnehmen kann, weshalb sich der Nutzer auf vier Ports pro Switch beschränken sollte. Der Simulator kann zwar mehr verwalten, jedoch überlagern sie sich dann in der Anzeige. Des Weiteren müssten dann einige Einstellungen bei den Timeout-Werten vorgenommen werden, damit die beiden Spanning Tree Algorithmen noch mit der großen Nachrichtenflut zurechtkommen. Jeder Port verfügt wie auch die Switches über eine Reihe von Attributen, die bei seiner Erzeugung initialisiert werden müssen. Hinzu kommen noch vier Methoden, die das Senden der Nachrichten regeln. Das Empfangen findet in dem Interface Connector statt.

Neben einer kleinen Anzahl an Attributen haben die Hosts zwei wichtige Methoden, die dem Erzeugen und Verschicken von Nachrichten dienen.

Die letzte größere Klasse sind die Wires. Sie repräsentieren die Verbindungslinien zwischen zwei Ports bzw. einem Port und einem Host. Jede Wire verfügt immer über genau einen Anfang und ein Ende. Das bedeutet, dass es keine Verzweigungen innerhalb eine Leitung gibt. Auch darf jeder Host und jeder Port nur exakt an einer Wire angeschlossen sein.

Zum Schluss kommen noch die beiden Enumerations „Portart“ und „Portstatus“, die beide ausschließlich von der Port-Klasse verwendet werden. Sie enthalten Informationen, die nur für die beiden implementierten Spanning Tree Algorithmen von Interesse sind.

Bisher wurde erklärt, woraus ein Netzwerk besteht und dass es vier verschiedene Nachrichtenarten geben soll. In Abbildung 28 sind die Beziehungen zwischen den Algorithmen und diesen Nachrichten veranschaulicht.

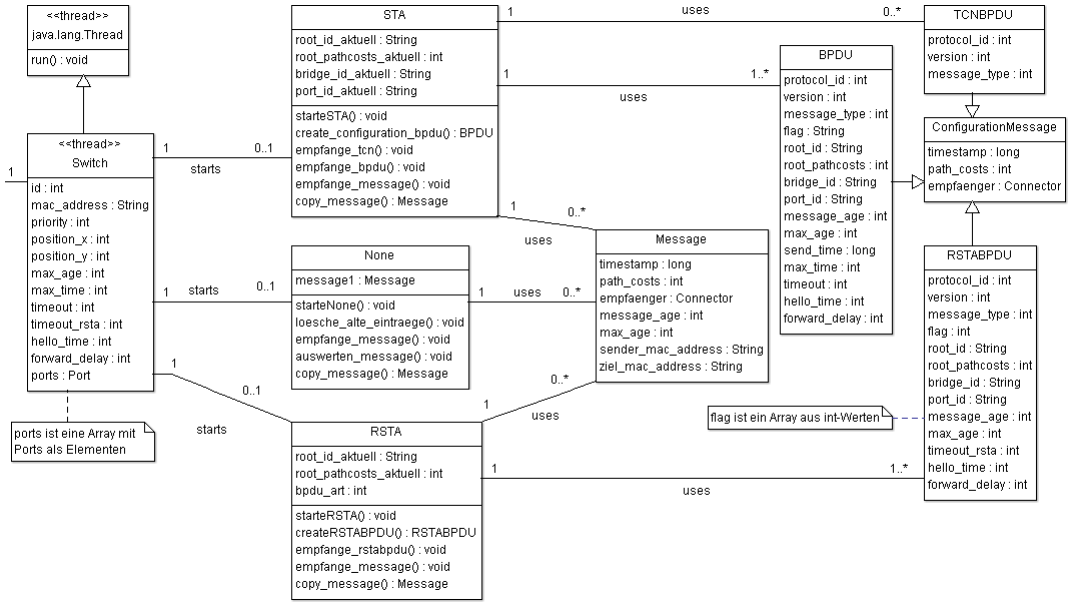


Abbildung 28: Klassendiagramm von den Nachrichten

Anders als in der Bachelorarbeit [Jan10b] haben die Switches selbst keine steuernde Funktion mehr. Das Einzige, was sie jetzt noch machen, ist das Starten des jeweiligen Algorithmus. In den Klassen STA und RSTA befinden sich die gesamten Implementationen des Spanning Tree Algorithmus IEEE 802.1D und des Rapid Spanning Tree Algorithmus. Sie sind die ausführenden Klassen, die Nachrichten erzeugen und auswerten. Der erstgenannte Algorithmus benutzt zum Aufbau und zur Rekonfiguration des Spannbaums BPDUs und TCNBPDUs. Der Rapid Spanning Tree Algorithmus verwendet dazu nur die RSTABPDUs. All diese drei BPDU-Typen sind von der Klasse „ConfigurationMessage“ abgeleitet, die die Attribute beinhaltet, die von allen drei BPDUs benötigt werden.

In der Klasse None ist die Möglichkeit implementiert ein Netzwerk ohne Spanning Tree Algorithmus laufen zu lassen. In diesem Fall kann der Nutzer nur Messages von einem Host zu einem anderen verschicken. Diese Nachrichten können aber unabhängig vom gewählten Algorithmus immer versendet werden.

Das Verschicken von BPDUs im Netzwerk funktioniert vereinfacht dargestellt wie folgt:

1. Switch erzeugt Nachricht
2. Switch übergibt Nachricht an Port
3. Port gibt Nachricht an Wire
4. Wire leitet Nachricht an gegenüberliegenden Port weiter
5. Gegenüberliegender Port empfängt Nachricht
6. Port gibt Nachricht an Switch
7. Switch wertet Nachricht aus
8. Switch leitet Nachricht weiter

Das Verschicken von Messages verläuft im Prinzip genauso. Der einzige Unterschied ist der, dass ein Host diese Nachrichten erzeugt und sie an die angeschlossene Wire übergibt. Beim Empfang einer Message geht es genau andersherum. [Jan10b]

4.2 Package „spanningTreeAlgorithm“

Dieses Package beinhaltet die folgenden Klassen:

- Main
- GUI
- Parser
- Unparser
- Configurations
- SAT_MAC_Table

Alle hier aufgeführten Klassen befassen sich ausschließlich mit dem Spanning Tree Simulator an sich und nicht mit den Algorithmen. Die Main-Klasse erzeugt eine Instanz der GUI-Klasse, wodurch das Hauptfenster des Simulators aufgerufen wird.

Die zentrale Klasse in diesem Package ist die GUI-Klasse. In ihr ist das Hauptfenster des Simulators und die Darstellung des Netzwerks implementiert. Auch entsteht hier das Fenster zur Anzeige der empfangenen Configuration Messages.

Der Parser dient dazu ein Configuration-File, in dem ein Netzwerk mittels XML definiert ist, in den Simulator zu laden. Der Unparser macht genau das Umgekehrte. Er setzt wieder aus den im Simulator vorhanden Informationen ein Configuration-File zusammen.

In der Configurations-Klasse wird das Fenster erzeugt, das die wichtigsten Attribut-Werte der einzelnen Netzwerkkomponenten darstellt. Des Weiteren kann der Nutzer hier einige davon nachträglich anpassen.

Die SAT-MAC-Table bietet dem Nutzer ein Fenster, in dem das Wissen des gewählten Switches über die Hosts angezeigt wird. In ihm ist zu sehen über welchen Port ein Host zu erreichen ist. [Jan10b]

5 Deaktivieren/Aktivieren einer Verbindungsleitung

Bevor auf die Rekonfiguration des Spannbaums beim Spanning Tree Algorithmus im Kapitel 6 eingegangen werden kann, muss erstmal die Möglichkeit erläutert werden, beliebige Verbindungsleitungen ausfallen lassen zu können.

5.1 Arbeitsweise des Deaktivierens/Aktivierens

Eine grundlegende Voraussetzung für die Rekonfiguration eines Spannbaums ist das Ausfallen einer Verbindungsleitung zwischen zwei Ports. Um dies zu simulieren, kann ein Nutzer während der laufenden Simulation über die Kontextmenü-Einträge „Deactivate Port“ bzw. „Activate Port“ jedes Ports die entsprechende Verbindungsleitung mit den beiden dazugehörigen Ports deaktivieren bzw. anschließend wieder aktivieren.

Sobald der Menüpunkt zum Deaktivieren einer Leitung betätigt wurde, wird diese sofort etwas breiter und rot eingefärbt. Dieser Zustand ist in Abbildung 29 dargestellt.

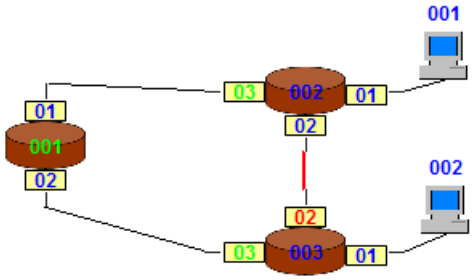


Abbildung 29: Deaktivieren einer Verbindungsleitung

In diesem Beispiel wurde die Leitung zwischen Switch 001 und 002 deaktiviert.

Zusätzlich zu der Funktionalität einzelne Leitungen zu deaktivieren und zu aktivieren, kann der Nutzer über die Kontextmenü-Einträge „Deactivate all Ports of Switch“ und „Activate all Ports of Switch“ alle Wires eines Switches gleichzeitig ausfallen lassen und anschließend wieder aktivieren. Dies ist in Abbildung 30 zu erkennen Wenn es bei einem Switch gleichzeitig aktivierte und deaktivierte Ports gibt, werden beide Menüeinträge angezeigt.

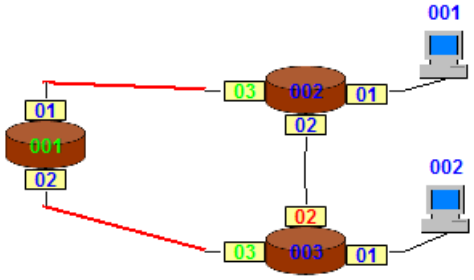


Abbildung 30: Deaktivieren aller Verbindungsleitungen

In diesem Beispiel wurden alle Leitungen des Switches 001 deaktiviert.

Durch das Deaktivieren einer Verbindungsleitung, die an einem Root-Port angeschlossen ist, tritt beim Switch nach Ablauf des Timeout-Intervalls ein Timeout ein, der zur Rekonfiguration des Spannbaums führt, wie in Kapitel 6 ausführlich erklärt wird.

5.2 Implementation des Deaktivierens/Aktivierens

Wie im vorherigen Unterkapitel beschrieben, kann der Nutzer beliebige Verbindungsleitungen ausfallen lassen. Der dazugehörige Quellcode befindet sich in Listing 1.

```
public static void deaktivierenActionPerformed(
      java.awt.event.ActionEvent evt, JMenuItem item, Port aport) {
   Connector anderes_ende = aport.getOtherEnd();
   if (aport.test_deaktivierter_port == false) {
      item = new JMenuItem("Activate Port " + aport.label_port.getText().toString());
      aport.test_deaktivierter_port = true;
      aport.port_of_switch.allePortsAktiviert = false;
      if (Port.class.isInstance(anderes_ende)) {
         Port eport = ((Port) anderes_ende);
         eport.test_deaktivierter_port = true;
         eport.port_of_switch.allePortsAktiviert = false;
         testAllePorts(eport);
      }
   }
   else {
      item = new JMenuItem("Deactivate Port " + aport.label_port.getText().toString());
      aport.test_deaktivierter_port = false;
      if (Port.class.isInstance(anderes_ende)) {
         Port eport = ((Port) anderes_ende);
         eport.test_deaktivierter_port = false;
         eport.port_of_switch.allePortsDeaktiviert = false;
         testAllePorts(eport);
      }
   }
   testAllePorts(aport);
}
```

Listing 1: Deaktivieren/Aktivieren eines Ports

Wenn ein Nutzer den entsprechenden Kontextmenüpunkt eines Ports anklickt, wird sofort die Methode aus Listing 1 aufgerufen. Es wird zuerst geprüft, ob der Port noch aktiviert ist oder bereits deaktiviert wurde. In beiden Fällen wird der bisherige Menüpunkt durch einen neuen ersetzt, mit dessen Hilfe die gerade durchgeführte Aktion wieder umgekehrt werden kann. Anschließend wird dem Port und Switch mitgeteilt, welchen der beiden Zustände die gewählte Komponente nun bekommt. Danach wird auch der gegenüberliegende Port in denselben Zustand versetzt. Als Abschluss wird mit der Methode „testAllePorts()“ geprüft, ob nun alle Ports des gewählten und gegenüberliegenden Switches deaktiviert bzw. aktiviert sind. Die Notwendigkeit des Tests wird im nächsten Absatz deutlich.
Zusätzlich zu dieser gerade beschriebenen Möglichkeit Verbindungsleitungen zu deaktivieren oder zu aktivieren, kann der Nutzer über zwei separate Kontextmenüpunkte jedes Switches all seine angeschlossenen Leitungen gleichzeitig deaktivieren bzw. aktivieren. Welcher der beiden Menüeinträge sichtbar ist, wird dadurch bestimmt, wie viele Ports des jeweiligen Switches denselben Zustand aufweisen. Sollten alle aktiviert sein, ist nur der Eintrag zum Deaktivieren sichtbar. Im Fall, dass alle deaktiviert sind, ist nur der Eintrag zum Aktivieren vorhanden. In allen anderen Situationen sind beide Menüeinträge sichtbar.

Da es beim Switch zwei separate Menüeinträge gibt, sind auch zwei Methoden vorhanden, die im Listing 2 zu sehen sind.

```
public static void alleAktivierenActionPerformed(
      java.awt.event.ActionEvent evt, JMenuItem item, Switch aswitch) {
   for (int i = 0; i < aswitch.ports.length; i++) {
      Port dport = aswitch.ports[i];
      if (dport.test_deaktivierter_port == true) {
         dport.test_deaktivierter_port = false;
         Connector anderes_ende = dport.getOtherEnd();
         if (Port.class.isInstance(anderes_ende)) {
            ((Port) anderes_ende).test_deaktivierter_port = false;
            testAllePorts(((Port) anderes_ende));
         }
      }
   }
   aswitch.allePortsAktiviert = true;
   aswitch.allePortsDeaktiviert = false;
}
public static void alleDeaktivierenActionPerformed(
      java.awt.event.ActionEvent evt, JMenuItem item, Switch aswitch) {
   for (int i = 0; i < aswitch.ports.length; i++) {
      Port dport = aswitch.ports[i];
      if (dport.test_deaktivierter_port == false) {
         dport.test_deaktivierter_port = true;
         Connector anderes_ende = dport.getOtherEnd();
         if (Port.class.isInstance(anderes_ende)) {
            ((Port) anderes_ende).test_deaktivierter_port = true;
            testAllePorts(((Port) anderes_ende));
         }
      }
   }
   aswitch.allePortsDeaktiviert = true;
   aswitch.allePortsAktiviert = false;
}
```

Listing 2: Deaktivieren/Aktivieren aller Ports eines Switches

Beide in Listing 2 aufgeführten Methoden sind vollkommen analog. Deshalb wird hier nur auf die erste eingegangen. Um alle Verbindungsleitungen eines Switches gleichzeitig zu aktivieren, wird über alle seine Ports iteriert und geprüft, ob diese deaktiviert oder aktiviert sind. Bei denjenigen, die bereits aktiviert sind, geschieht nichts. Die anderen werden zusammen mit den jeweils gegenüberliegenden Ports aktiviert, damit die bisher ausgefallene Verbindungsleitung wieder genutzt werden kann. Dabei muss mit der Methode „testAllePorts()“ geprüft werden, ob beim gegenüberliegenden Switch nun ebenfalls alle Ports aktiviert sind. Warum dies getestet wird, wurde bereits direkt vor dem Listing 2 beschrieben. Am Ende der Methode, sind alle Ports des gewählten Switches aktiviert.

Bisher wurde ausführlich erklärt, wie Ports aktiviert bzw. deaktiviert werden können. Jedoch hat das Deaktivieren auch grafische Folgen, wie im vorherigen Unterkapitel beschrieben. Deaktivierte Leitungen werden rot eingefärbt und auch etwas dicker dargestellt. In Abbildung 29 ist solch eine deaktivierte Leitung zu sehen. Der dazugehörige Quellcode befindet sich in Listing 3.

```
protected void paintComponent(Graphics g) {
   super.paintComponent(g);
   Graphics2D g2 = (Graphics2D) g;
   Stroke stroke = new BasicStroke(1);
   g2.setColor(Color.black);
   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;
         if (Port.class.isInstance(end)) {
            Port end_port = (Port) end;
            if (end_port.test_deaktivierter_port == true) {
               stroke = new BasicStroke(2);
               g2.setColor(Color.red);
               g2.setStroke(stroke);
               g2.drawLine(start_port.position_x,
                     start_port.position_y, end_port.position_x,
                     end_port.position_y);
               panel_netzwerk.updateUI();

            } else {
               stroke = new BasicStroke(1);
               g2.setColor(Color.black);
               g2.setStroke(stroke);
               g2.drawLine(start_port.position_x,
                     start_port.position_y, end_port.position_x,
                     end_port.position_y);
               panel_netzwerk.updateUI();
            }
         }...}}}
```

Listing 3: Einfärben von deaktivierten Verbindungsleitungen

Die in Listing 3 aufgeführte Methode dient dem Erzeugen der Verbindungslinien in der grafischen Anzeige des Netzwerks, was bereits in der Bachelorarbeit [Jan10b] behandelt wurde. Deshalb wird sich hier nur auf die Linien zwischen zwei Ports beschränkt, da der Code für die Leitungen zwischen einem Port und einem Host analog ist. Zuallererst musste das Graphics-Objekt g zu einem Graphics2D-Objekt gecastet werden, da nur diese 2D-Objekte die Möglichkeit bieten die Linienstärke zu verändern. In der zweiten If-Verzweigung wird überprüft, ob das andere Ende der Verbindungsleitung auch ein Port ist. Sollte dies zutreffen, wird getestet, ob dieser vom Nutzer deaktiviert wurde. Wenn dem so ist, wird die Linie rot und mit der doppelten Linienstärke angezeigt. Sollte der Port aktiv sein, wird er schwarz und mit der normalen Linienstärke dargestellt. Bei einer Verbindungslinie zwischen einem Port und einem Host funktioniert es analog. [Jan10b]

6 Rekonfiguration des Spannbaums beim STA IEEE 802.1D

Dieses Kapitel besteht aus den zwei Unterkapiteln 6.1 und 6.2. Es befasst sich mit der Rekonfiguration des Spannbaums beim Spanning Tree Algorithmus IEEE 802.1D.
Dies ist die erste und wichtigste Erweiterung des Spanning Tree Simulators, da sie ein Bestandteil jedes Spanning Tree Algorithmus darstellt. Im ersten Unterkapitel wird zuerst ihre Funktionsweise anhand eines Beispiels ausführlich erklärt, um dann bei der Implementation zu verdeutlichen, wie diese Funktionalität im Simulator umgesetzt wurde.

6.1 Arbeitsweise der Rekonfiguration

In Kapitel 2.2 wurde bereits die grundlegende Funktionsweise des Spanning Tree Algorithmus IEEE 802.1D anschaulich erklärt. Jedoch wurde dabei die Rekonfiguration des Spannbaums ausgelassen. Hier folgt nun eine detailreiche Beschreibung der eben erwähnten Funktionalität, die mit mehreren Abbildungen veranschaulicht wird.
Die Ausgangssituation ist in Abbildung 31 zu sehen und zeigt das in dem „Configuration-File 01“ definierte Netzwerk.

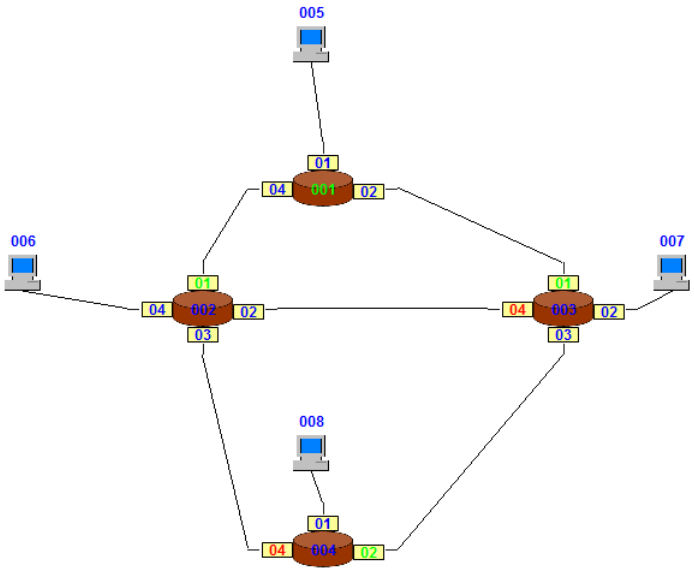


Abbildung 31: Configuration-File 01 mit fertigem Spannbaum

Wie in Abbildung 31 zu erkennen ist, hat der Spanning Tree Algorithmus den Spannbaum bereits vollständig aufgebaut und das gesamte Netzwerk befindet sich in einem stabilen Zustand. Damit ist gemeint, dass es sich ohne äußere Einflüsse nicht mehr verändern wird. In diesem Beispiel ist „Switch 001“ der Root-Switch und die Verbindungsleitungen, die mit einem der beiden rot markierten und somit blockierten Ports verbunden sind, werden nicht genutzt, da es sonst Schleifen im Netz geben würde. Wie der Algorithmus solch einen Spannbaum aufbaut, wurde bereits im Kapitel 2.2 erläutert, weshalb hier nicht erneut auf diesen Vorgang eingegangen wird.

Ein Ziel dieser Masterarbeit ist die Rekonfiguration des Spannbaums nach dem Ausfall einer Verbindungsleitung zwischen zwei Switches. Da sich der Algorithmus nach dem Aufbau dieses Baums in einem stabilen Zustand befindet, war es erforderlich den Ausfall einer Leitung zu simulieren. Deshalb wurde dem Nutzer die Möglichkeit geboten, manuell beliebige Verbindungsleitungen ausfallen lassen zu können. Dies ist über das Kontextmenü des entsprechenden Ports möglich. Diese Funktionalität ist jedoch nur während der laufenden Simulation möglich. Wer einen Port schon vor dem Starten der Simulation deaktivieren möchte, kann dies über den Menüpunkt „Configurations“ des jeweiligen Ports tun. Dort kann der Port auf den Status „Disabled“ gesetzt werden. In diesem Fall ist der Port zwar deaktiviert, aber die Leitung selbst ist noch aktiv. Um die Simulation möglichst realistisch zu gestalten, wurden beide Möglichkeiten realisiert.[Hei07, Hei09, Jan10b, Ras08]

6.1.1 Ausfall einer Verbindungsleitung

Klickt der Anwender den Menüpunkt „Deactivate Port“ an, werden der gewählte Port, die angeschlossene Verbindungsleitung sowie der gegenüberliegende Port deaktiviert. Dieser Zustand wird durch das rote Einfärben der jeweiligen Leitung signalisiert.

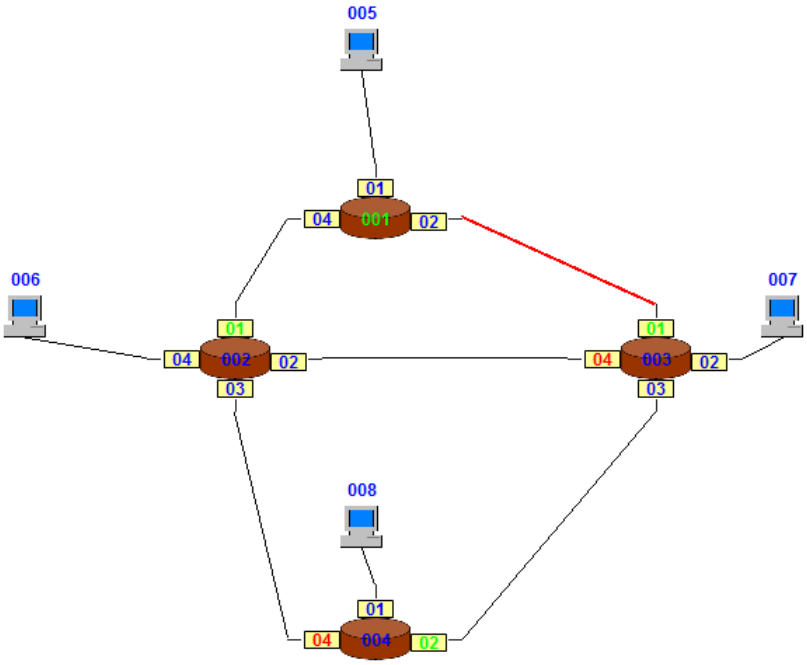


Abbildung 32: Configuration-File 01 mit deaktivierter Verbindungsleitung

Wie in Abbildung 32 zu erkennen ist, wurde die Verbindungsleitung zwischen „Switch 001“ und „Switch 003“ deaktiviert. Da „Switch 003“ nun über seinen Root-Port keine BPDUs mehr vom Root-Switch empfängt, bekommt dieser nach Ablauf des Timeout-Intervalls ein Timeout. Im hier verwendeten Beispiel liegt es bei 20 Sekunden.

Als erste Folge des Timeouts vergisst der Switch sein gesamtes Wissen über den Root-Switch, also wer dieser war und mit welchen Pfadkosten er erreichbar war. Danach geht er davon aus, dass er selbst der Root-Switch ist, und wandelt all seine Ports wieder zu Designated-Ports um. Dieser Zustand wird in Abbildung 33 veranschaulicht.

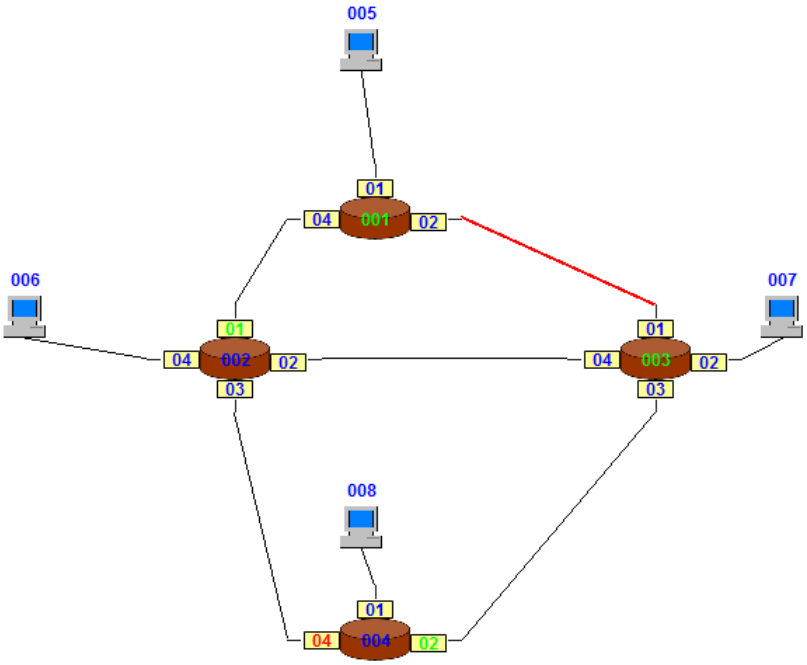


Abbildung 33: Configuration-File 01 mit Timeout bei Switch 003

Nun schickt „Switch 003“ mittels Broadcast eine neue Sorte von Nachrichten an all seine Nachbarn, und zwar die „Topology-Change-Notification-BPDUs“ oder kurz TCN-BPDUs. Diese sind deutlich kürzer als die normalen BPDUs und enthalten keinerlei Topologie-Informationen.

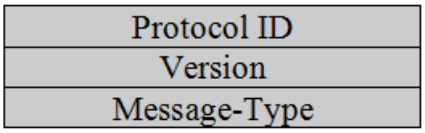
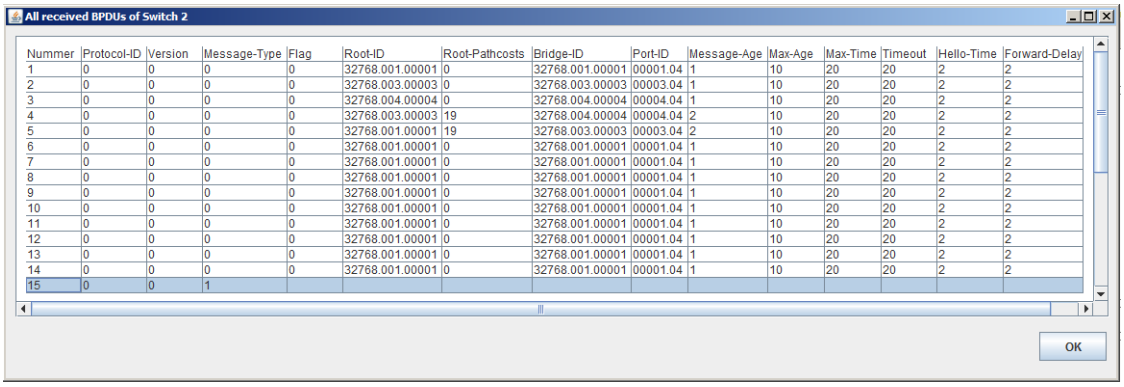


Abbildung 34: Schematischer Aufbau einer TCN-BPDU

In der Abbildung 34 wird der schematische Aufbau einer solchen TCN-BPDU dargestellt. Wie zu sehen ist, besteht diese nur aus den ersten drei Feldern einer normalen BPDU, und zwar aus der „Protocol ID“, der „Version“ und dem „Message-Type“. Die ersten beiden Werte bleiben auch bei diesem Nachrichtentyp auf 0 gesetzt. Jedoch das Feld „Message-Typ“ bekommt dieses Mal eine 1, da die 0 für die normalen BPDUs reserviert ist.

Als zusätzliche Veranschaulichung folgt die Abbildung 35. Sie zeigt in der blau markierten Zeile die TCN-BPDU, die von „Switch 003“ verschickt und von „Switch 002“ empfangen wurde.



Nummer	Protocol-ID	Version	Message-Type	Flag	Root-ID	Root-Pathcosts	Bridge-ID	Port-ID	Message-Age	Max-Age	Max-Time	Timeout	Hello-Time	Forward-Delay
1	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
2	0	0	0	0	32768.003.00003	0	32768.003.00003	00003.04	1	10	20	20	2	2
3	0	0	0	0	32768.004.00004	0	32768.004.00004	00004.04	1	10	20	20	2	2
4	0	0	0	0	32768.003.00003	19	32768.004.00004	00004.04	2	10	20	20	2	2
5	0	0	0	0	32768.001.00001	19	32768.003.00003	00003.04	2	10	20	20	2	2
6	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
7	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
8	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
9	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
10	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
11	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
12	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
13	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
14	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
15	0	0	1											

Abbildung 35: Eine empfangene TCN-BPDU

Nachdem der „Switch 003“ mittels Broadcast jeweils eine TCN-BPDU an all seine Nachbar-Switches geschickt hat, startet wiederum ein Timeout-Intervall, das aber diesmal die doppelte Länge wie das ursprüngliche Intervall hat. Nun wartet der „Switch 003“ auf eine Antwort-BPDU, die fast genauso aufgebaut ist wie eine normale BPDU. Was es damit auf sich hat, wird etwas weiter unten erklärt. Erst wenn ein Switch, bei dem ein Timeout eingetreten ist, eine solche Antwort erhält, nimmt dieser seine Tätigkeit als richtiger Root-Switch auf. Bis dahin wartet er auf eine Antwort–BPDU und macht sonst nichts. Was geschieht, wenn innerhalb des Timeout-Intervalls keine Antwort eintrifft, wird ebenfalls etwas weiter unten beschrieben.
Irgendwann empfangen „Switch 002“ und „Switch 004“ die verschickten TCN-BPDUs. Sobald ein Switch, der kein aktiver Root-Switch ist und sich auch nicht in einer Timeout-Situation befindet, eine solche Nachricht empfängt, leitet er diese einfach weiter. Sollte sich der Empfänger in einer Timeout-Situation befinden, ignoriert er diese Nachricht. Wie der aktive Root-Switch auf solch eine Nachricht reagiert, wird erst etwas weiter unten erklärt.
Empfängt ein normaler Switch, der die eben erwähnten Bedingungen erfüllt, eine TCN-BPDU, sind zwei Fälle zu unterscheiden, und zwar je nachdem, ob er diese Nachricht über den Root-Port empfangen hat, oder nicht. Wie in Abbildung 33 zu sehen ist, liegt bei „Switch 002“ der erste und bei „Switch 004“ der zweite Fall vor. „Switch 002“ leitet die empfangene TCN-BPDU nur über seinen Root-Port, also direkt auf dem kürzesten Weg zum Root-Switch, weiter. Dies kann „Switch 004“ jedoch nicht tun, da er die Nachricht über den bisher kürzesten Weg bekommen hat. Also leitet er die Nachricht über all seine anderen Ports weiter. Wie sich anhand dieser Beschreibung vermuten lässt, ist das Ziel jeder TCN-BPDU der Root-Switch, da in der Regel nur er Antwort-BPDUs erzeugen und verschicken kann.
Nachdem die beiden Switches die TCN-BPDUs weitergeleitet haben, vergessen sie wie „Switch 003“ ihr gesamtes Wissen über den Root-Switch, wandeln alle Ports in Designated-Ports um und gehen davon aus, dass sie selbst der Root-Switch sind. Nun starten auch sie ein neues Timeout-Intervall und warten auf eine Antwort-BPDU.

Sollte der bisherige Root-Switch noch über irgendeinen Weg erreichbar sein, wird er früher oder später eine TCN-BPDU empfangen. Bei so kleinen Netzwerken wie das vom „Configuration-File 01“ befinden sich irgendwann alle Switches, bis auf den echten Root-Switch im Wartemodus. Diese Situation ist in Abbildung 36 zu sehen.

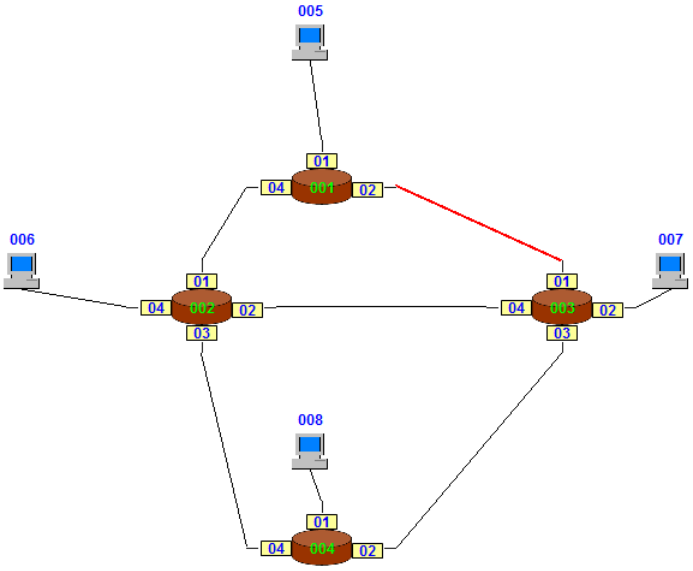


Abbildung 36: Configuration-File 01 mit wartenden Switches

Da nun der Root-Switch eine TCN-BPDU empfangen hat, erzeugt und verschickt er nun mittels Broadcast Antwort-BPDUs. Diese sind fast identisch mit den normalen BPDUs. Sie unterscheiden sich nur in einem Punkt, und zwar, dass sie im Feld „Flag“ eine 1 anstatt einer 0 stehen haben. Die Abbildung 37 zeigt in der blau markierten Zeile die Antwort-BPDU, die vom Root-Switch verschickt und von „Switch 002“ empfangen wurde.

All received BPDUs of Switch 2

Nummer	Protocol-ID	Version	Message-Type	Flag	Root-ID	Root-Pathcosts	Bridge-ID	Port-ID	Message-Age	Max-Age	Max-Time	Timeout	Hello-Time	Forward-Delay
1	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
2	0	0	0	0	32768.003.00003	0	32768.003.00003	00003.04	1	10	20	20	2	2
3	0	0	0	0	32768.004.00004	0	32768.004.00004	00004.04	1	10	20	20	2	2
4	0	0	0	0	32768.001.00001	19	32768.003.00003	00003.04	2	10	20	20	2	2
5	0	0	0	0	32768.003.00003	19	32768.004.00004	00004.04	2	10	20	20	2	2
6	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
7	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
8	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
9	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
10	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
11	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
12	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
13	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2
14	0	0	1											
15	0	0	1											
16	0	0	0	1	32768.001.00001	0	32768.001.00001	00001.04	1	10	20	20	2	2

OK

Abbildung 37: Eine empfangene Antwort-BPDU

Sobald ein Switch eine solche BPDU erhält, wertet er diese ganz normal aus, und zwar egal, in welchem Zustand sich der Empfänger gerade befindet. Danach leitet er diese Antwort wie jede andere BPDU auch an seine Nachbarn weiter. Auf diese Weise verbreitet sich das Wissen über das gesamte Netzwerk und ein neuer Spannbaum wird aufgebaut. Der neue fertige Spannbaum ist in Abbildung 38 zu erkennen.

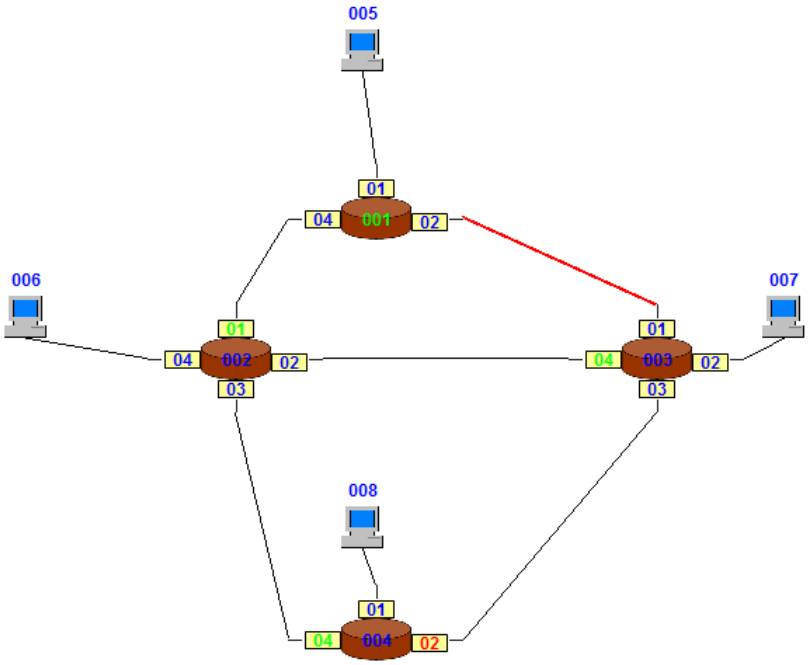


Abbildung 38: Configuration-File 01 mit neuem Spannbaum

In dem hier beschriebenen Beispiel ist zwar eine Verbindungsleitung ausgefallen, jedoch war der Root-Switch noch über einen anderen Weg zu erreichen. Wie die Rekonfiguration arbeitet, wenn „Switch 001“ überhaupt nicht mehr zu erreichen ist, wird jetzt erklärt.[Hei07, Hei09, Jan10b, Ras08]

6.1.2 Ausfall der letzten Verbindungsleitung zum Root-Switch

Nun wird die Verbindungsleitung zwischen „Switch 001“ und „Switch 002“ deaktiviert. Dies ist in Abbildung 39 visualisiert.

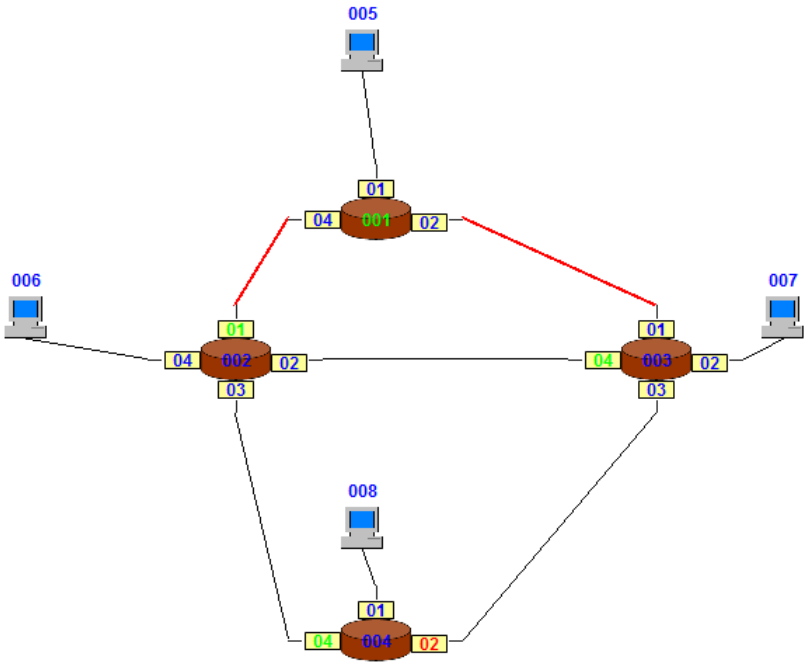


Abbildung 39: Configuration-File 01 mit nicht mehr erreichbarem Root-Switch

Bei einem der Switches wird irgendwann wieder ein Timeout auftreten. Im Folgenden wird davon ausgegangen, dass bei „Switch 002“ als Erstes ein Timeout eingetreten ist. Er verfährt genau so, wie es eben schon beschrieben wurde. Auch „Switch 003“ und „Switch 004“, die die TCN-BPDUs empfangen, reagieren gleich. Irgendwann wird wieder der Zustand in Abbildung 36 erreicht. Der einzige Unterschied ist, dass der Root-Switch diesmal keine Möglichkeit hat, eine TCN-BPDU zu empfangen bzw. eine Antwort-BPDU zu verschicken. Da alle drei Switches auf eine Antwort warten, die sie nie bekommen werden, wird irgendwann bei einem von ihnen ein erneutes Timeout eintreten. In diesem Fall verlässt der entsprechende Switch den Wartemodus, erzeugt und verschickt selbst mittels Broadcast Antwort-BPDUs. Die Switches, die solch eine Nachricht empfangen, werten diese wie gewohnt aus und nach kurzer Zeit etabliert sich auch hier ein neuer Spannbaum.

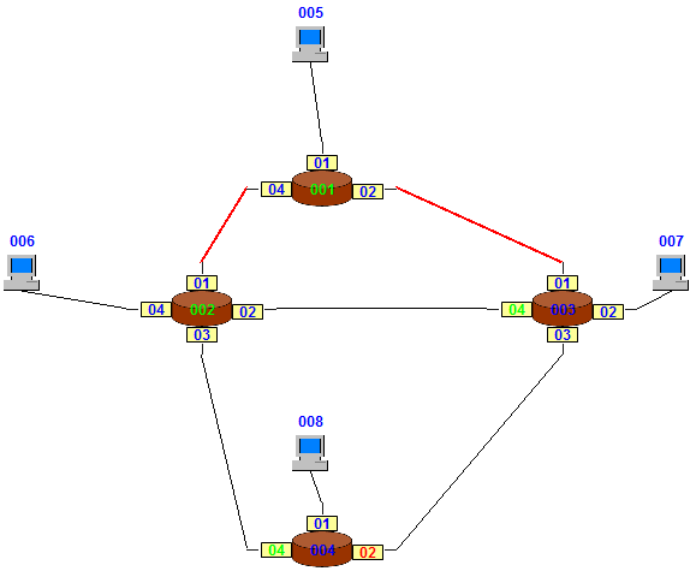


Abbildung 40: Configuration-File 01 mit neuem Spannbaum

Wie in Abbildung 40 zu erkennen ist, hat sich auch im zweiten beschriebenen Fall ein Spannbaum gebildet. Jedoch gibt es nun zwei Root-Switches und zwar der nicht mehr erreichbare „Switch 001“ und der neue Wurzelknoten „Switch 002“.

Sollte eine Verbindungsleitung deaktiviert werden, die geblockt und somit nicht genutzt wird, passiert nichts, da bei keinem der Switches ein Timeout eintritt. [Hei07, Hei09, Jan10b, Ras08]

6.1.3 Aktivierung einer Verbindungsleitung

Die beiden beschriebenen Beispiele behandeln nur Fälle, in denen eine oder auch mehrere Verbindungsleitungen zwischen jeweils zwei Switches ausfallen. Was passiert aber, wenn keine Leitung ausfällt, sondern eine aktiviert wird? In einem solchen Fall empfängt ein Switch über einen neuen Weg eine normale BPDU, wertet diese wie üblich aus und blockiert gegebenenfalls einen Port. Diese Funktionalität bietet aber bereits der ganz normale Spanning Tree Algorithmus ohne Rekonfiguration.

Ein prinzipielles Problem bei der Rekonfiguration stellt das alte Wissen dar. Sobald ein Switch durch ein eingetretenes Timeout feststellt, dass er den Root-Switch nicht mehr erreichen kann, informiert er die anderen Switches, wie zuvor beschrieben, darüber und es wird ein neuer Spannbaum aufgebaut. Bis alle Switches über den Ausfall informiert wurden, sind immer noch BPDUs mit veraltetem Wissen unterwegs. Diese stellen eine ernste Gefahr für die erfolgreiche Rekonfiguration des Spannbaums dar, da sie unter Umständen endlos im Netz kursieren könnten. Deshalb werden alle BPDUs bei der Erzeugung mit einer zusätzlichen Altersinformation versehen. Sollte ein Switch eine zu alte BPDU empfangen, ignoriert er diese einfach. [Hei07, Hei09, Jan10b, Ras08]

6.2 Implementation der Rekonfiguration

Da nun die Funktionsweise der Rekonfiguration des Spannbaums bekannt ist, wird in diesem Unterkapitel auf deren Realisierung im Spanning Tree Simulator eingegangen.

6.2.1 TCN-BPDU

Als Erstes wird hier der neue Nachrichtentyp TCN-BPDU vorgestellt, dessen Aufbau schon im vorherigen Unterkapitel beschrieben wurde. In der zugrunde liegenden Bachelorarbeit [Jan10b] gab es, abgesehen von den Nutznachrichten, die sich die Hosts gegenseitig zuschicken, nur einen Nachrichtentyp, und zwar die normale BPDU. Bei der Rekonfiguration wird aber ein zweiter BPDU-Typ benötigt. Deshalb wurde die Oberklasse „ConfiguarionMessage“ erstellt, von der die beiden Klassen „BPDU“ und „TCNBPDU“ abgeleitet sind.

TCN-BPDUs sind deutlich kleiner als die des anderen Typs, da sie keinerlei Topologie-Informationen oder Nutzdaten enthalten. Anders als bei normalen BPDUs enthält das Feld „message_type“ diesmal anstatt einer 0 eine 1. Dies signalisiert dem Empfänger, dass die empfangene Nachricht eine TCN-BPDU ist und mindestens eine Leitung im Netzwerk ausgefallen sein muss. Was dies zur Folge hat, wurde bereits im Unterkapitel 6.1 erschöpfend erklärt. Sobald der sendende Port die Nachricht auf die Wire schickt, leitet sie diese direkt an den Empfänger-Port weiter, der die BPDU in die Empfangsliste inbox1 des jeweiligen Switches einfügt. Um die Zeitverzögerung durch die Pfadkosten einer Leitung zu realisieren, wird jede BPDU mit einer Empfangszeit versehen, nach der sie in die Empfangsliste einsortiert wird. Erst wenn die vom Nutzer eingestellte Verzögerung abgelaufen ist, wird sie vom Switch aus der Liste herausgeholt und ausgewertet. [Hei07, Hei09, Jan10b, Ras08]

6.2.2 Alterung der BPDUs

Ein weiterer Aspekt der Rekonfiguration ist die Alterung der BPDUs. Bisher wurde das Alter einer Nachricht nur mittels der zurückgelegten Strecke im Netz, also der Anzahl der Switches, die diese weitergeleitet haben, bestimmt. Dies ist eine Möglichkeit dafür zu sorgen, dass Nachrichten in Netzwerken, in denen Schleifen vorhanden sind, nicht endlos weitergeleitet werden. Jedoch reicht diese Alterung bei kleinen Netzwerken, die eine Baumstruktur aufweisen, nicht aus, da diese Sicherheitsvorkehrung erst viel zu spät oder gar nicht greift. Deshalb wurde, wie es beispielsweise die Quelle [Hei07] fordert, bei der Rekonfiguration auch die zeitliche Alterung realisiert. Jede BPDU bekommt bei der Erzeugung durch den Root-Switch einen Zeitstempel hinzugefügt. Dies ist eine Variable, in der die bei der Erzeugung aktuelle Systemzeit gespeichert wird. Das maximal erlaubte Alter wird vom Nutzer bei der Erstellung eines Netzwerks in dem Configuration-File angegeben, kann aber auch noch im Simulator verändert werden. Da das Alter nur als eine einfache Variable dargestellt wird, findet der entscheidende Prozess der Alterung beim Empfang einer BPDU statt. Dort muss einmal geprüft werden, um welche Art von BPDU es sich handelt und ob diese zu alt ist. Im Listing 4 ist dieser Entscheidungsprozess zu sehen.

```
if (!inbox1.isEmpty()) {
   ConfigurationMessage cm;
   try {
      cm = inbox1.take();
      if (BPDU.class.isInstance(cm)) {
         bpdu2 = (BPDU) cm;
         cm = null;
         while (System.currentTimeMillis() > bpdu2.send_time + max_time * 1000
               || bpdu2.message_age >= bpdu2.max_age) {
            if (!inbox1.isEmpty()) {
               try {
                  cm = inbox1.take();
                  if (BPDU.class.isInstance(cm)) {
                     bpdu2 = (BPDU) cm;
                     cm = null;
                  } else {
                     bpdu2 = null;
                     break;
                  }
               } catch (InterruptedException e) {
                  e.printStackTrace();
               }
            }
            else {
               bpdu2 = null;
               break;
            }}}
```

Listing 4: Alterung der BPDUs

Nachdem geprüft wurde, ob die „PriorityBlockingQueue inbox1“, in der alle empfangenen BPDUs gespeichert werden, nicht leer ist, wird das erste Element aus dieser genommen und gleichzeitig entfernt. Danach wird, wie im Listing 4 zu erkennen, geschaut, was für eine BPDU vorliegt. Sollte es eine normale sein, wird mittels einer While-Schleife geprüft, ob die Nachricht zu alt ist. Sollte dies zutreffen, wird einfach die nächste BPDU aus der Liste genommen und erneut auf

ihren Typ getestet. Auf diese Weise werden alle normalen BPDUs, die zu alt sind, übersprungen. Sobald eine Nachricht gefunden wurde, die sich noch im erlaubten Zeitrahmen befindet, oder die Liste leer ist, wird die Schleife sofort verlassen. Zusammengefasst bedeutet dies, dass die erste TCN-BPDU oder die erste nicht zu alte normale BPDU aus der Liste genommen wird.

Dabei gibt es jedoch eine Gefahr. Wenn ein Switch schneller Nachrichten empfängt, als er diese auswerten kann, wächst die Liste inbox1 immer weiter. Dies führt dazu, dass die wartenden Nachrichten irgendwann zu alt werden. Wie eben beschrieben, werden zu alte BPDUs einfach gelöscht. Wenn aber zu viele veralte in der Liste sind, kann der Switch nie wieder aktuelle Nachrichten entnehmen, da die Altersprüfung und die anschließende Löschung auch Zeit in Anspruch nehmen. Dies führt dazu, dass der Switch nie wieder die While-Schleife verlässt, keine neuen Nachrichten mehr auswertet und dann ein Timeout eintritt. Ob so ein Fall eingetreten ist, kann der Nutzer daran erkennen, dass bei betroffenen Switches am laufenden Band Timeouts auftreten, obwohl ihre Entfernung zum Root-Switch gar nicht so groß ist. Wenn so eine Situation eintritt, sollten am besten ein paar Verbindungsleitungen des betroffenen Switches mittels „Disabled“ deaktiviert werden, oder der Anwender erhöht die Hello-Time, was zur Folge hat, dass der Root-Switch seltener BPDUs verschickt. Bei der zweiten Lösung kann es aber erforderlich sein, das Timeout-Intervall ebenfalls zu verlängern. [Hei07, Hei09, Jan10b, Ras08]

6.2.3 Empfang einer normalen und einer Antwort-BPDU

In Listing 5 ist zu lesen, was mit einer empfangenen normalen BPDU geschieht, die noch nicht zu alt ist.

```
        if (bpdu2 != null) {
           if (bpdu2.flag.equals("1") && test_time_out == true) {
              synchronized (inbox2) {
                 inbox2.add(bpdu2);
              }
              aport = (Port) bpdu2.empfaenger;
              aport.status = Portstatus.LISTENING;
              empfange_bpdu();
              if (isInterrupted() == true) {
                 return;
              }
           }
           else if (bpdu2.flag.equals("0") && test_time_out == false) {
              synchronized (inbox2) {
                 inbox2.add(bpdu2);
              }
              aport = (Port) bpdu2.empfaenger;
              aport.status = Portstatus.LISTENING;
              empfange_bpdu();
              if (isInterrupted() == true) {
                 return;
              }
           }
           else {
              bpdu2 = null;
           }}
```

Listing 5: Empfang einer normalen BPDU

Hier sind drei Fälle und zwei BPDU-Typen zu unterscheiden. Die beiden Typen sind zum einen die Antwort-BPDUs, die vom Root-Switch nur versendet werden, falls ein Timeout eingetreten ist und zum anderen die normalen BPDUs, die regelmäßig vom Root-Switch verschickt werden.

Zuerst wird der zweite Fall betrachtet, da dieser die Standard-Situation darstellt. Eine solche Situation ist gegeben, falls noch kein Timeout eingetreten und eine normale BPDU eingetroffen ist. Als Erstes wird die Nachricht in der „ArrayList inbox2“ abgelegt. Dort werden alle empfangenen und ausgewerteten BPDUs unabhängig ihres Typs gespeichert, um vom Nutzer abrufbar zu sein. Anschließend wechselt der Empfänger-Port in den Listening-Status und ruft die Methode „empfange_bpdu()“ auf, die die Nachricht dann endgültig auswertet.

Im ersten Fall ist schon ein Timeout eingetreten und der Switch, der auf eine Antwort-BPDU wartet, empfängt diese auch. Mit solch einer Antwort-BPDU wird genauso verfahren wie in der gerade eben beschriebenen Situation. Die Auswertung einer solchen Nachricht ist in Listing 11 zu sehen.

Der letzte Fall tritt in zwei Situationen ein, und zwar einmal, wenn bei dem Empfänger-Switch noch kein Timeout eingetreten ist, dieser aber trotzdem eine Antwort-BPDU empfängt und

zweitens, wenn bei dem Empfänger-Switch ein Timeout eingetreten ist, er aber eine normale BPDU in der Liste vorfindet. In beiden Situationen wird die entsprechende BPDU einfach gelöscht und ignoriert.[Hei07, Hei09, Jan10b, Ras08]

6.2.4 Empfang einer TCN-BPDU

Wie mit einer TCN-BPDU umgegangen wird, ist in Listing 6 beschrieben.

```
if (cm != null) {
   if (id == 1) {
   }
   synchronized (inbox2) {
      inbox2.add(cm);
   }
   TCNBPDU tcn = (TCNBPDU) cm;
   aport = (Port) tcn.empfaenger;
   aport.status = Portstatus.LISTENING;
   empfange_tcn();
}
```

Listing 6: Empfang einer TCN-BPDU

Wenn eine TCN-BPDU in der Liste inbox1 gefunden wurde, wird diese ebenfalls zuerst in die „ArrayList inbox2“ kopiert. Danach geht der Empfänger-Port in den Listening-Status, um die Methode „empfange_tcn()“ aufzurufen. Diese ist in Listing 9 abgebildet.[Hei07, Hei09, Jan10b, Ras08]

6.2.5 Zurücksetzen des Timeout-Timers

Bevor jedoch die Auswertung vorgestellt wird, ist es sinnvoll erst einmal auf die Erstellung solch einer Nachricht einzugehen. Wie im Unterkapitel 6.1 beschrieben, werden TCN-BPDUs nur verschickt, falls bei einem Switch ein Timeout eingetreten ist. Dieses tritt aber nur ein, wenn ein Switch keine BPDUs mehr vom Root-Switch über seinen Root-Port empfängt. In dieser Situation wird der Timeout-Timer nicht zurückgesetzt und es tritt eine Zeitüberschreitung des Timeout-Intervalls ein.

```
if (aport.art == Portart.ROOT_PORT && bpdu2.root_id.equals(root_id_aktuell)
      || bpdu2.flag.equals("1")) {
   last_received_root_bpdu = System.currentTimeMillis();
}
```

Listing 7: Zurücksetzen des Timout-Intervalls

Wie in Listing 7 zu erkennen ist, wird der Timeout-Timer eines Switches zurückgesetzt, indem in die Variable „last_received_root_bpdu“ einfach die aktuelle Systemzeit eingetragen wird. Dies wird in zwei Fällen getan, und zwar einmal, wenn bei einem Switch noch kein Timeout eingetreten ist und dieser über seinen Root-Port eine gültige normale BPDU vom Root-Switch empfängt und zweitens, wenn bei einem Switch ein Timeout eingetreten ist und bei ihm eine Antwort-BPDU eintrifft. Wie lang das Timeout-Intervall sein soll, legt der Nutzer beim Erstellen eines Netzwerks in dem Configuration-File fest. Er kann es aber auch im Simulator mittels des Menüpunkts „Configurations“ ändern.

Wie im Unterkapitel 6.1 beschrieben, hat sich bei der Implementation herausgestellt, dass sich der vom Spanning Tree Algorithmus vorgesehene Zeitraum für das Timeout-Intervall nicht realisieren lässt, da der Simulator zu langsam ist. Warum dies so ist, liegt unter anderem an der getroffenen Entscheidung jeden Switch als einen eigenen Thread zu realisieren. Dies führt schon bei recht kleinen Netzwerken zu einer größeren Beanspruchung der Systemressourcen, was schlussendlich Zeit kostet.

Der Grund für diese Entscheidung war, dass das Verhalten von Netzwerken möglichst realistisch nachempfunden werden sollte. Da diese Masterarbeit auf die Bachelorarbeit [Jan10b] aufbaut, wurde die getroffene Entscheidung beibehalten. Mittels mehrerer Tests wurde für die vorliegenden Configuration-Files ein Intervall gefunden, bei dem der Nutzer nicht all zulange warten muss bis ein Timeout eintritt, der fertige Spannbaum aber dennoch stabil ist. Dieses beträgt abhängig davon, für welche der vorliegenden Configuration-Files sich der Nutzer entscheidet, 20-30 Sekunden.[Hei07, Hei09, Jan10b, Ras08]

6.2.6 Eintritt des ersten Timeouts

Nachdem nun geklärt wurde, wann der Timeout-Timer zurückgesetzt wird, kann der Fall betrachtet werden, was passiert, wenn bei einem Switch ein Timeout eingetreten ist. Dieser wird in Listing 8 beschrieben.

```
if (GUI.checkbox_reconfiguration.isSelected() == true) {
   if (root_switch == false && test_time_out == false
         && System.currentTimeMillis() > last_received_root_bpdu + timeout * 1000) {
      test_time_out = true;
      root_switch = true;
      erster_durchlauf = true;
      inbox1.clear();
      last_received_root_bpdu = System.currentTimeMillis();
      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());
      for (int i = 0; i < ports.length; i++) {
         ports[i].root_id_zwischen = "0";
         ports[i].root_pathcosts_zwischen = 0;
      }
      for (int i = 0; i < ports.length; i++) {
         ports[i].art = Portart.DESIGNATED_PORT;
         ports[i].status = Portstatus.FORWARDING;
         TCNBPDU tcn1 = new TCNBPDU();
         ports[i].sendTCN(tcn1);
      }}
```

Listing 8: Eingetretenes Timeout

Die erste If-Verzweigung des Listings 8 dient der Kontrolle, ob der Nutzer mittels der Checkbox im Hauptfenster des Simulators die Rekonfiguration an- oder abgeschaltet hat. Standardmäßig ist sie angeschaltet. In der zweiten müssen drei Bedingungen überprüft werden, und zwar, ob der aktuelle Switch kein Root-Switch ist und er sich bisher noch nicht in einer Timeout-Situation befindet, aber gerade ein Timeout eingetreten ist. Die erste Bedingung ist wichtig, da ein Root-Switch erst einmal kein Timeout bekommen kann, dieser aber, weil er in der Regel keine BPDUs empfängt, eintreten würde. Mit der zweiten Bedingung ist gemeint, dass der Switch nicht auf eine Antwort-BPDU warten darf. Bei der dritten wird geschaut, ob das Timeout-Intervall überschritten wurde.
Sollten alle drei Bedingungen zutreffen, geht der Switch davon aus, dass er selbst der Root-Switch ist. Er vergisst darum sein gesamtes Wissen über den bisherigen Root-Switch, also wer dieser war und mit welchen Pfadkosten dieser erreicht werden konnte. Dann löscht er die Liste inbox1 mit allen Nachrichten. Dies ist wichtig, damit das nun veraltete Wissen nicht mehr im Netz verbreitet werden kann. Nachdem der Timeout-Timer zurückgesetzt wurde, trägt er in die beiden

nun leeren Variablen „root_id_aktuell“ und „root_pathcosts_aktuell“, die das Wissen über den Root-Switch repräsentieren, seine ID und bei den Pfadkosten 0 ein. Anschließend wird auch das Wissen der Ports gelöscht. Zum Abschluss erzeugt und verschickt der Switch mittels Broadcast eine TCN-BPDU an alle seine Nachbarn. Bei diesem letzten Schritt wandelt er alle Ports in Designated-Ports um. [Hei07, Hei09, Jan10b, Ras08]

6.2.7 Auswerten einer TCN-BPDU

Nun sind die TCN-BPDUs unterwegs und werden irgendwann von anderen Switches im Netzwerk empfangen. Dabei sind zwei Fälle zu unterscheiden, und zwar, dass der Empfänger ein normaler Switch oder ein Root-Switch ist. Der erste Fall ist in Listing 9 zu finden.

Zu Beginn wird geprüft, ob der Empfänger kein Root-Switch ist. Sollte dies zutreffen, geht er davon aus, dass er nun der Root-Switch ist, löscht die Liste inbox1, vergisst sein Wissen über den bisherigen Root-Switch und trägt in die entsprechenden Variablen seine eigenen Werte ein. Also verfährt ein Switch beim Eintreten eines Timeouts und beim Empfang einer TCN-BPDU sehr ähnlich. Der einzige Unterschied befindet sich in der If-Bedingung in der unteren Hälfte des Listings 9. Diese Bedingung behandelt das Weiterleiten einer TCN-BPDU. Wenn ein Switch solch eine Nachricht über seinen Root-Port empfängt, bedeutet dies, dass auch seine Verbindung zum Root-Switch unterbrochen ist. Deshalb schickt er die Nachricht über all seine andren Ports weiter und wandelt alle Ports in Designated-Ports um. Sollte ein Switch eine TCN-BPDU über einen anderen Port als den Root-Port empfangen, schickt der die Nachricht nur über seinen Root-Port weiter. Auch in diesem Fall wandelt er all seine Ports in Designated-Ports um.

```
if (root_switch == false) {
   test_time_out = true;
   root_switch = true;
   erster_durchlauf = true;
   inbox1.clear();
   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());
   for (int i = 0; i < ports.length; i++) {
      ports[i].root_id_zwischen = "0";
      ports[i].root_pathcosts_zwischen = 0;
   }
   if (aport.art == Portart.ROOT_PORT) {
      for (int i = 0; i < ports.length; i++) {
         if (aport.id != ports[i].id) {
            ports[i].art = Portart.DESIGNATED_PORT;
            ports[i].status = Portstatus.FORWARDING;
            TCNBPDU tcn1 = new TCNBPDU();
            ports[i].sendTCN(tcn1);
         } else {
            ports[i].art = Portart.DESIGNATED_PORT;
            ports[i].status = Portstatus.FORWARDING;
         }
      }
   }
   else {
      for (int i = 0; i < ports.length; i++) {
         if (ports[i].art == Portart.ROOT_PORT) {
            ports[i].art = Portart.DESIGNATED_PORT;
            ports[i].status = Portstatus.FORWARDING;
            TCNBPDU tcn1 = new TCNBPDU();
            ports[i].sendTCN(tcn1);
         } else {
            ports[i].art = Portart.DESIGNATED_PORT;
            ports[i].status = Portstatus.FORWARDING;
         }}}}
```

Listing 9: Auswerten einer TCN-BPDU wenn Switch kein Root-Switch ist

Sollte der Empfänger-Switch ein Root-Switch sein, muss wieder zwischen zwei Fällen unterschieden werden, und zwar, ob er der bisherige Root-Switch ist, oder ob der Switch wegen eines eingetretenen Timeouts auf eine Antwort-BPDU wartet. Diese beiden Situationen sind in Listing 10 zu sehen.

```
else {
   if (test_time_out == false) {
      BPDU bpdu1;
      for (int i = 0; i < ports.length; i++) {
         ports[i].status = Portstatus.FORWARDING;
         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 = "1";
            bport.sendBPDU(bpdu1);
            last_send_time = System.currentTimeMillis();
         }
      }
   }
   else {
      for (int i = 0; i < ports.length; i++) {
         ports[i].art = Portart.DESIGNATED_PORT;
         ports[i].status = Portstatus.FORWARDING;
      }}}
```

Listing 10: Auswerten einer TCN-BPDU wenn Switch ein Root-Switch ist

In der If-Bedingung aus Listing 10 wird zuerst der erste Fall betrachtet. Wenn also der echte Root-Switch eine TCN-BPDU empfängt, erzeugt und verschickt er mittels Broadcast jeweils eine Antwort-BPDU über all seine Ports. Diese unterscheidet sich nur im Feld Flag von einer normalen BPDU, da sie dort eine 1 und keine 0 aufweist.
Im zweiten Fall, in dem der Switch nur auf eine Antwort-BPDU wartet, ignoriert er die empfangene TCN-BPDU und wandelt seine Ports nochmals in Designated-Ports um.[Hei07, Hei09, Jan10b, Ras08]

6.2.8 Auswerten einer Antwort-BPDU

Der Empfang einer Antwort-BPDU ist in Listing 11 zu erkennen.

```
if (bpdu2.flag.equals("1")) {
   test_time_out = false;
   inbox1.clear();
}
...
else {
BPDU bpdu1;
for (int i = 0; i < ports.length; i++) {
   if (ports[i].art == Portart.DESIGNATED_PORT) {
      if (ports[i].status == Portstatus.FORWARDING
            || ports[i].status == Portstatus.LEARNING) {
         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 = "1";
            bpdu1.send_time = bpdu2.send_time;
            cport.sendBPDU(bpdu1);
         }}}}}
```

Listing 11: Empfang einer Antwort-BPDU

Die Auswertungen einer Antwort-BPDU und einer normalen BPDU verlaufen fast identisch. Falls ein Switch, der sich in einer Timeout-Situation befindet und somit auf eine Antwort-BPDU wartet, eine solche empfängt, löscht er nochmals den Inhalt der Liste inbox1 und verlässt diesen Zustand. Das bedeutet, dass er jetzt ein ganz normaler Root-Switch ist und sich wie ein solcher verhält. Er wird entsprechend der Hello-Time normale BPDUs erzeugen und verschicken. Auf diese Weise handeln die einzelnen Switches des Netzwerks einen neuen Spannbaum aus.
Bevor der Switch seine Tätigkeit als Root-Switch aufnimmt, schickt er die empfangene Antwort-BPDU an all seine Nachbarn weiter.
An der Stelle im Listing 11, wo sich die drei Punkte befinden, findet die eigentliche Auswertung der Nachricht statt. Diese unterscheidet sich aber nicht von der einer normalen BPDU und kann in der Bachelorarbeit [Jan10b] nachgelesen werden. [Hei07, Hei09, Jan10b, Ras08]

6.2.9 Ausbleiben einer Antwort-BPDU

Nun fehlt nur noch ein Aspekt im Rahmen der Rekonfiguration, und zwar was passiert, wenn die Antwort-BPDU bei einem wartenden Switch nicht rechtzeitig eintrifft. Diese Situation ist in Listing 12 zu erkennen.

```
else if (test_time_out == true
      && System.currentTimeMillis() > last_received_root_bpdu
            + timeout * 1000 * 2) {
   inbox1.clear();
   last_received_root_bpdu = System.currentTimeMillis();
   test_time_out = false;
   BPDU bpdu1;
   for (int i = 0; i < ports.length; i++) {
      ports[i].status = Portstatus.FORWARDING;
      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 = "1";
         bport.sendBPDU(bpdu1);
         last_send_time = System.currentTimeMillis();
      }}}
```

Listing 12: Nicht rechtzeitig eingetroffene Antwort-BPDU

Mögliche Gründe für das Ausbleiben einer Antwort-BPDU sind einmal, dass die Antwort zu lange braucht, um anzukommen, oder zweitens, wenn der bisherige Root-Switch über keinen Weg mehr erreichbar ist und somit auch nie eine Antwort-BPDU verschickt hat.
Falls bei einem Switch ein Timeout eintritt, weil er vom Root-Switch keine normale BPDU mehr empfängt, wird der Timeout-Timer zurückgesetzt. Sollte nun innerhalb des doppelten Zeitraums keine Antwort-BPDU ankommen, tritt beim Switch ein zweites Timeout auf. Für das zweite Intervall musste ein längerer Zeitraum verwendet werden, da die verschickten TCN-BPDUs den Root-Switch erreichen und dann die Antwort-BPDUs zurück gelangen müssen.
Nachdem ein zweites Timeout eingetreten ist, löscht der Switch den Inhalt der Liste inbox1. Dann setzt er den Timeout-Timer zurück, verlässt den Wartemodus, erzeugt und verschickt mittels Broadcast Antwort-BPDUs an all seine Nachbarn. Danach wird er zu einem echten Root-Switch und verschickt entsprechend der Hello-Time normale BPDUs. Auf diese Weise erstellen die Switches im Netzwerk einen neuen Spannbaum.
Sollte das zweite Timeout eingetreten sein, weil der bisherige Root-Switch nicht mehr erreichbar ist, wird natürlich einer der anderen Switches ein echter Root-Switch bleiben. Das bedeutet, dass aus ursprünglich einem Netzwerk zwei oder mehr unabhängige Netze entstehen können und es in jedem dieser neuen Teilnetze einen eigenen Root-Switch gibt. [Hei07, Hei09, Jan10b, Ras08]

7 Implementation des Rapid Spanning Tree Algorithmus

Zusätzlich zum ursprünglichen Spanning Tree Simulator wurde der Spanning Tree Simulator um den Rapid Spanning Tree Algorithmus erweitert. Da nun beide Algorithmen im Simulator enthalten sind, kann der Nutzer beide nacheinander auf demselben Netzwerk laufen lassen. Auf diese Weise werden die Unterschiede zwischen beiden Verfahren am schnellsten sichtbar.
Um die Verständlichkeit zu erhöhen, wird dieses Kapitel in zwei Unterkapitel unterteilt, in denen getrennt voneinander der Aufbau des Spannbaums und die Rekonfiguration dieses erklärt werden. Wie in den vorherigen Kapiteln dieser Arbeit werden auch hier die Funktionsweise und die Implementation des Algorithmus in separaten Unterunterkapiteln erläutert.

7.1 Aufbau des Spannbaums beim RSTA

Als erster Aspekt des Rapid Spanning Tree Algorithmus wird in diesem Unterkapitel beschrieben, wie der Spannbaum anhand des eben erwähnten Algorithmus aufgebaut wird. Das grundlegende Wissen, also zum Beispiel wie die BPDUs aufgebaut sind, welche Portarten und Portstatus es gibt, wurde bereits im Unterkapitel 2.3 ausführlich erklärt.

7.1.1 Arbeitsweise des Spannbaumaufbaus beim RSTA

Um den Rapid Spanning Tree Algorithmus starten zu können, muss entweder ein Netzwerk mittels Configuration-File ins Programm geladen werden oder es muss ein neues Netz direkt im Simulator erstellt werden. Beim Start werden alle Switch-Threads initialisiert. Die Hauptaufgaben beider Spanning Tree Protokolle ist es das aktuelle Netzwerk, das als Graph vorliegt, in eine Baumstruktur zu überführen. Dazu muss einmal der Wurzelknoten also der Root-Switch bestimmt, alle Schleifen erkannt und diese dann aufgebrochen werden. Auch die Vorgehensweise ist bei beiden Protokollen recht ähnlich.
Der ursprüngliche Spanning Tree Algorithmus hat aber einen großen Nachteil, der beim neuen behoben wurde. Nachdem der alte erkannt hat, dass eine Leitung ausgefallen ist, braucht er unter Umständen bis zu 30 Sekunden, um den Spannbaum vollständig zu rekonfigurieren. Während dieser Zeit werden keinerlei Host-Nachrichten übertragen, was bedeutet, dass das Netzwerk für bis zu 30 Sekunden nicht nutzbar ist. Diese Zeitspanne ist bei heutigen Netzwerken einfach zu lang. Deshalb wurde der Rapid Spanning Tree Algorithmus entwickelt, der den Spannbaum deutlich schneller aufbaut.

Auch beim Rapid Spanning Tree Algorithmus arbeitet jeder Switch als selbstständige Komponente, was zur Folge hat, dass es kein globales Wissen über das gesamte Netzwerk gibt und niemand die Kontrolle darüber hat. Also hat jeder Switch zu Beginn nur sein privates Wissen und muss mit den anderen mittels BPDUs kommunizieren, um neues Wissen zu erlangen. Dies ist unter anderem zur Bestimmung des Root-Switches zwingend erforderlich.

Nachdem nun ein Netzwerk in dem Simulator angezeigt wird und der Algorithmus gestartet wurde, geht jeder Switch zuerst einmal davon aus, dass er selbst der Root-Switch ist, da er nur über sein privates Wissen verfügt. Dann bestimmt er seine Edge-Ports, da er dafür keinerlei weiteres

Wissen benötigt. Sollte solch ein Port irgendwann einmal eine BPDU empfangen, wird er sofort zu einem Designated-Port und bekommt den Discarding-Status. Dies kann jedoch nur vorkommen, wenn die Verbindung zum Host während des laufenden Algorithmus zu einem Switch umgelegt wird. Da dies im Simulator nicht möglich ist, wurde der Aspekt auch nicht implementiert. Alle anderen Ports wandelt der Switch zu Designated-Ports um und setzt sie in den Discarding-Status. Nun befindet sich jeder Switch in der initialen Ausgangslage, die in Abbildung 41 dargestellt ist.

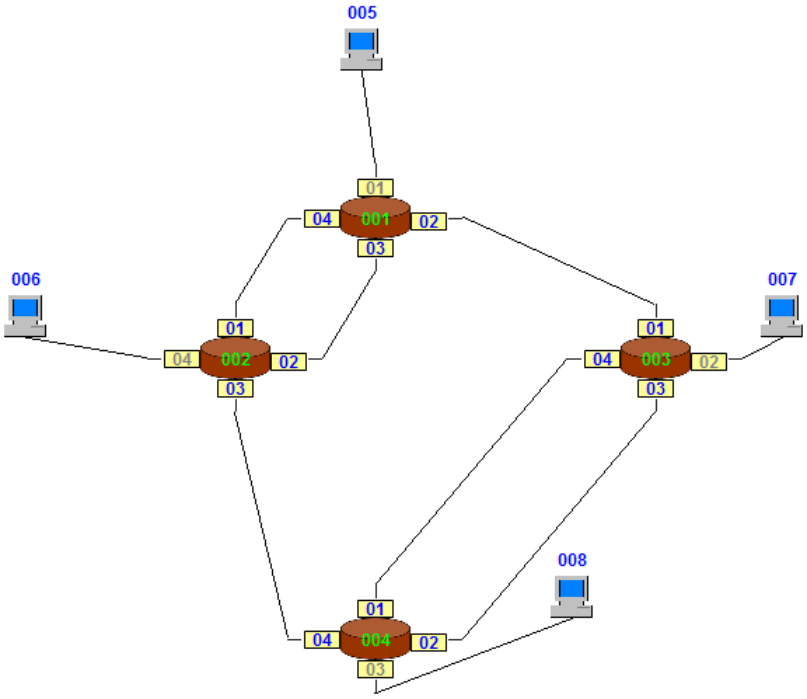


Abbildung 41: Ausgangssituation eines Netzwerks nach dem Starten des RSTA

Sobald bei einem Switch ein Designated-Port im Discarding-Status vorliegt, wird über diesen eine Proposal-BPDU an den benachbarten Switch geschickt. In Abbildung 41 trifft dies auf alle blau eingefärbten Ports zu. Diese Nachricht ist eine ganz normale BPDU, in der das Proposal-Flag auf 1 gesetzt wird (im Unterunterkapitel 2.3.1 nachzulesen). Die einzelnen Felder der BPDU werden genau wie im ursprünglichen Spanning Tree Algorithmus mit dem privaten Wissen des sendenden Switches ausgefüllt (im Unterkapitel 2.2 und in Quelle [Jan10b] nachzulesen). Anschließend begibt sich der Sender in einen Wartemodus, den er erst durch Eintreffen der entsprechenden Agreement-BPDU oder nach Ablauf eines Timers wieder verlässt.
Das Timer-Intervall soll im Idealfall der dreifachen Hello-Time entsprechen, was leider im Simulator nicht realisierbar war. Wie bereits mehrfach erwähnt, handelt es sich bei dieser Arbeit nur um eine Simulation eines Netzwerks, was bedeutet, dass alle Netzwerkkomponenten sich ein und dieselben Systemressourcen teilen müssen. Deshalb müssen die Timer-Intervalle entsprechend angepasst werden. Die siebenfache Hello-Time hat sich im vorliegenden Netzwerk als ein guter und stabiler Wert herauskristallisiert.
Solange sich ein Switch in diesem Wartemodus befindet, ignoriert er alle normalen BPDUs und Host-Nachrichten. Proposal-, Agreement und Topology-Change-BPDUs empfängt er weiterhin problemlos.

Sobald ein Switch eine Proposal-BPDU empfängt, wertet er sie ganz normal aus. In Abbildung 42 sind die drei Proposal-BPDUs zu sehen, die Switch 001 empfängt.

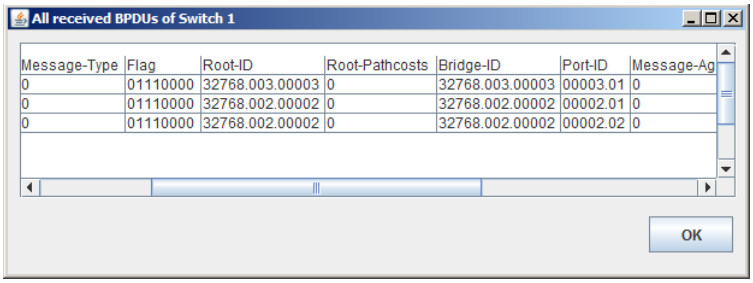


Abbildung 42: Die drei empfangenen Proposal-BPDUs von Switch 001

Wie zu erkennen ist, wurde die Abbildung 42 nur auf den relevanten Teil der Nachrichten beschränkt, da der Screenshot ansonsten zu groß geworden wäre. Da das zweite Bit des Flag-Felds auf 1 gesetzt ist, handelt es sich hierbei zweifelsfrei um drei Proposal-BPDUs. Die erste stammt von Switch 003 und die beiden anderen von Switch 002, was im Feld „Bridge-ID“ und „Port-ID“ steht. An den Feldern „Root-ID“ und „Root-Pathcosts“ kann abgelesen werden, dass beide Sender davon ausgehen, dass sie selbst der Root-Switch sind.

Nach dem Empfang solch einer Proposal-BPDU versetzt der Switch all seine anderen Ports außer die Edge-Ports in den Discarding-Status. Beim erstmaligen Empfang solch einer Nachricht ändert sich dadurch nichts, da sich alle Ports noch in der Ausgangssituation befinden. Im späteren Verlauf kann dies anders aussehen. Anschließend wird der Empfänger-Port in den Learning-Status gesetzt und er merkt sich das empfangene Wissen, also die Root-ID, die Root-Pathcosts, die Bridge-ID und die Portart des Senders, die im Flag-Feld verschlüsselt enthalten ist. Bevor der Switch beginnt, das Wissen mit seinem eigenen zu vergleichen, erstellt er eine Kopie der Nachricht, setzt anstatt dem Proposal-Bit das Agreement-Bit und schickt sie an den Sender zurück.

Nun beginnt, wie eben angekündigt, der Switch damit sein bisheriges Wissen mit dem empfangenen zu vergleichen, um herauszubekommen, ob er überhaupt der echte Root-Switch ist.
Dazu werden die folgenden Werte der beiden Switches der Reihe nach überprüft:

- Priorität
- ID
- Mac-Adresse

Sollte der auswertende Switch einen kleineren Wert haben, bleibt er Root-Switch, sollte er einen größeren haben, geht er davon aus, dass der andere der Root-Switch ist und sollten beide Werte gleich sein, wird das nächste Kriterium in der Reihe geprüft. Spätestens bei der Mac-Adresse kann beim ersten Auswerten einer BPDU eine eindeutige Entscheidung getroffen werden. Sollte herauskommen, dass der Sender der bessere Root-Switch ist, überschreibt der Empfänger-Switch sein privates Wisse also die Root-ID und die Root-Pathcosts mit dem neuen und wandelt den aktuellen Port in den Root-Port um.

Wenn der Fall eintritt, dass alle drei Kriterien gleich sind, bedeutet dies, dass der aktuelle Switch schon weiß wer der richtige Root-Switch ist. Dann muss nur noch geprüft werden, ob der Port, über den die aktuelle Nachricht eingetroffen ist, vielleicht eine bessere Verbindung zum Root-Switch hat, als der bisherige Root-Port.
Dafür werden diese Werte der beiden Ports der Reihe nach geprüft:

- Root-Pathcosts
- Priorität
- ID

Auch hier wird genauso verfahren wie eben. Der Port mit den kleineren Werten wird zum neuen Root-Port und der alte wird wieder zum Designated-Port. Wenn bei der Auswertung herauskommt, dass der aktuelle Switch der Root-Switch bleibt, macht er nichts. Auf dieselbe Weise verfährt der Switch auch mit normalen BPDUs, die aber erst nach Eintreffen der Agreement-BPDU verschickt werden. Nun wurde der Root-Switch und der Root-Port bestimmt. Diese Situation ist in Abbildung 43 dargestellt.

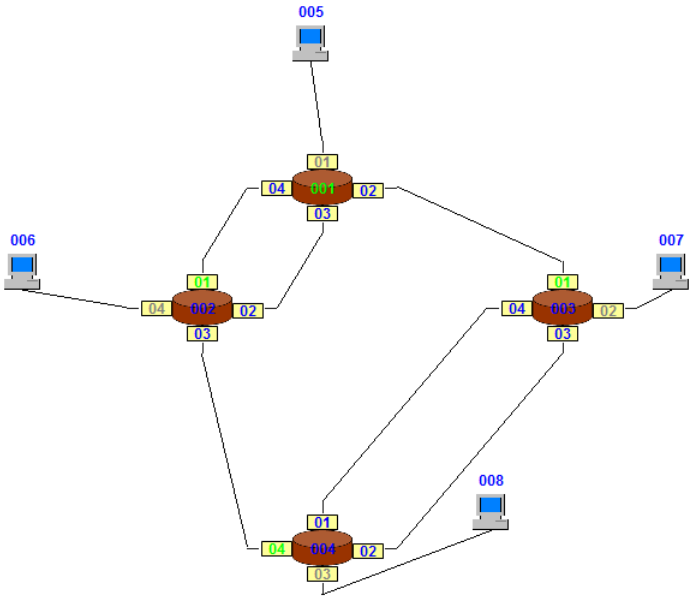


Abbildung 43: Das Netzwerk nach Auswertung der Proposal-BPDUs

Was hier vielleicht verwundert, ist, dass bei Switch 004 der Port 04 und nicht Port 01 der Root-Port ist, obwohl dieser die kleinere ID hat. Dies kommt, weil Port 04 absichtlich mit einem niedrigeren Wert bei der Priorität versehen wurde. Da dieses Kriterium vor der ID geprüft wird und somit wichtiger ist, liegt der Root-Port an dieser Stelle.

Sollte der Empfangsport jedoch kein Root-Port werden, muss geprüft werden, ob er vielleicht ein Backup- oder ein Alternate-Port ist. Nähere Informationen zu diesen beiden Portarten sind im Unterunterkapitel 2.3.3 zu finden. Als Erstes wird geschaut, ob der aktuelle Port zum selben

Switch wie der Root-Port führt. Sollte dem so sein, muss mit den bekannten Kriterien herausgefunden werden, welches Ende der Leitung zum Backup-Port gemacht wird. Wenn der aktuelle Port zu einem anderen Switch führt, er aber weiß wer der aktuelle Root-Switch ist, befindet er sich auf der Liste der möglichen Kandidaten zum Alternate-Port. Um endgültig zu bestimmen, ob die aktuelle Leitung eine Alternate-Leitung ist, müssen alle anderen Ports, die zum selben Switch führen, gefunden und getestet werden, ob der aktuelle der beste ist. Die dazu benötigten Kriterien stehen im Unterunterkapitel 2.3.3. Zu beachten ist hier noch, dass bei keiner der gefunden Leitungen ein Root-Port angeschlossenen sein darf. Es kann vorkommen, dass eine dieser Leitungen die Root-Leitung vom Nachbar-Switch ist. In diesem Fall darf natürlich keiner der Leitungen zur Alternate-Leitung werden, da sie alle Backup-Leitungen sind. Wenn die aktuelle die beste ist, muss wieder das richtige Ende gefunden werden, das zum Alternate-Port wird. Sollte sie nicht die beste sein, wird geprüft, welches Ende zum Backup-Port wird.

Auf diese Weise werden alle normalen und Proposal-BPDUs ausgewertet. Die Informationen in Agreement- und Topology-Change-BPDUs werden überhaupt nicht ausgewertet. Wenn die verschickte Agreement-BPDU innerhalb der siebenfachen Hello-Time beim entsprechenden Switch ankommt, geht dieser aus dem Wartemodus raus und versetzt den Empfangsport in den Forwarding-Status. Solch eine Nachricht ist in Abbildung 44 zu sehen.

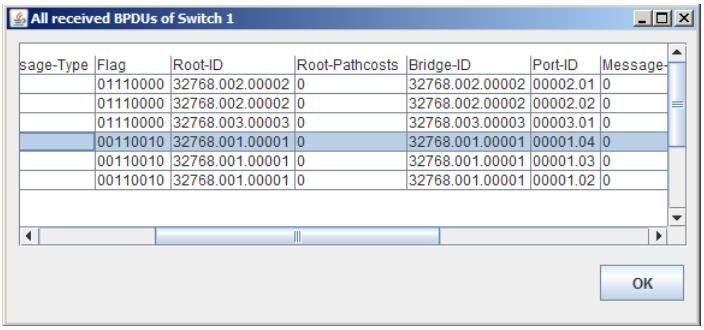


Abbildung 44: Die drei empfangenen Agreement-BPDUs von Switch 001

Wie in Abbildung 42 wird hier nur der wesentliche Teil der BPDUs gezeigt. An der gesetzten 1 bei Bit 7 des Flag-Felds können die Agreement-BPDUs erkannt werden. Da dieser BPDU-Typ nur eine Kopie einer vorher verschickten Proposal-BPDU ist, sind auch die Inhalte der restlichen Felder zu verstehen. Im ersten Moment sieht es danach aus, als ob sich Switch 001 selbst die drei BPDUs geschickt hätte, da in den Feldern Bridge-ID und Port-ID seine eigenen Werte stehen. Aber dies ist nicht der Fall, da sie Agreement-BPDUs sind.

Wenn solch eine BPDU nicht innerhalb des definierten Zeitintervalls eintrifft, wird der Port wie beim ursprünglichen Spanning Tree Algorithmus mittels der Forward-Delay-Timer in den Forwarding-Status gebracht. Weiterführende Informationen zu diesem Timer sind im Unterkapitel 2.2 und in der Quelle [Jan10b] zu finden.

Nachdem ein Port die Agreement-BPDU empfangen hat, verschickt der Switch entsprechend der Hello-Time normale BPDUs über alle aktiven Ports außer über die Edge-Ports. Erst mit den normalen BPDUs verbreitet sich das Wissen vom Root-Switch über das gesamte Netzwerk. Da beim Rapid Spanning Tree Algorithmus keine BPDUs mehr über die Nachbar-Switches hinaus weitergeleitet werden, reichen die Proposal-BPDUs nicht aus, um den Spannbaum vollständig aufzubauen. Dazu werden auch die normalen BPDUs benötigt. Schritt für Schritt werden so alle Schleifen erkannt und aufgebrochen, bis der schleifenfreie Spannbaum herauskommt. Diese Situation ist in Abbildung 45 dargestellt.

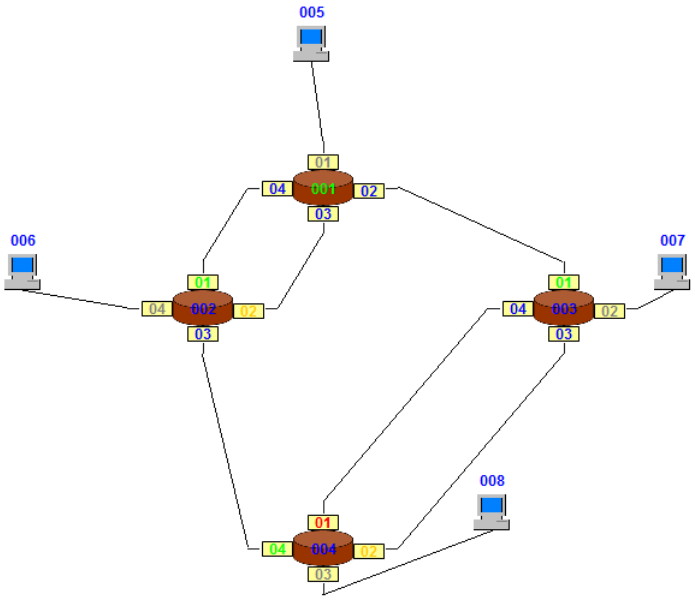


Abbildung 45: Der fertige Rapid Spanning Tree

Der fertige Spannbaum in Abbildung 45 bleibt ohne Beeinflussung durch den Nutzer so bestehen und kann jetzt Host-Nachrichten weiterleiten. Wie der Rapid Spanning Tree Algorithmus mit dem Ausfall einer Leitung umgeht, wird in Unterkapitel 7.2 beschrieben. [Wik11, Ava10, Bad05, Cis06, eTu12, Jan10b]

7.1.2 Implementation des Spannbaumaufbaus beim RSTA

Im vorherigen Unterunterkapitel wurde der Aufbau des Spannbaums mittels des Rapid Spanning Tree Algorithmus beschrieben. Hier wird jetzt erklärt, wie diese Funktionalität implementiert wurde.
Nach dem Start des Algorithmus gehen alle Switches davon aus, dass sie selbst der Root-Switch sind, bestimmen die Edge-Ports und wandeln alle anderen Ports zu Designated-Ports um. In Listing 13 ist zu sehen, wie die Ports in die Ausgangssituation versetzt werden.

```
for (int i = 0; i < ports.length; i++) {
   // Designated-Ports
   if (Port.class.isInstance(ports[i].getOtherEnd())) {
      ports[i].art = Portart.DESIGNATED_PORT;
      ports[i].status = Portstatus.DISCARDING;
      ports[i].label_port.setForeground(Color.BLUE);
   }
   // Edge-Ports
   else if (Host.class.isInstance(ports[i].getOtherEnd())) {
      ports[i].art = Portart.EDGE_PORT;
      ports[i].status = Portstatus.FORWARDING;
      ports[i].label_port.setForeground(Color.GRAY);
   }
}
```

Listing 13: Ausgangssituation der Ports

In den If-Bedingungen wird geprüft, ob der Port mit einem anderen Port oder mit einem Host verbunden ist. Im ersten Fall wird er zum Designated-Port und bekommt den Status Discarding. Im zweiten wird er zum Edge-Port und erhält den Status Forwarding, damit er die Host-Nachrichten weiterleiten kann.

Anschließend werden über alle Designated-Ports, die sich im Discarding-Status befinden, jeweils eine Proposal-BPDU verschickt. Es wird mit einer If-Bedingung getestet, welche Ports sich in der eben genannten Ausgangssituation befinden. Dann wird eine entsprechende BPDU erzeugt und mittels der „sendRSTABPDU()-Methode“ jedes Ports an die Wire übergeben. Das einzige Relevante ist hier die Verschlüsselung der Portart des sendenden Ports im Flag-Feld. Dies ist in Listing 14 dargestellt. Für die anderen Bits wird mittels einfacher If-Bedingungen geprüft, ob die Voraussetzungen erfüllt sind, um sie auf 1 zu setzen. Die Portarten Designated- und Root-Port haben eine eigene Bit-Folge, die in der ersten Hälfte des Listings zu erkenne ist. Die Bit-Folge 01 wird hingegen von zwei Portarten und zwar vom Alternate- und vom Backup-Port verwendet. Deshalb ist es nicht möglich diese beim Auswerten einer BPDU zu unterscheiden. Die letzte Verschlüsselung wurde für Ports mit unbekannter Portart aufgehoben.

```
...
if (aport.art == Portart.DESIGNATED_PORT) {
   rstabpdu1.flag[2] = 1;
   rstabpdu1.flag[3] = 1;
} else if (aport.art == Portart.ROOT_PORT) {
   rstabpdu1.flag[2] = 1;
   rstabpdu1.flag[3] = 0;
} else if (aport.art == Portart.ALTERNATE_PORT
      || aport.art == Portart.BACKUP_PORT) {
   rstabpdu1.flag[2] = 0;
   rstabpdu1.flag[3] = 1;
} else {
   rstabpdu1.flag[2] = 0;
   rstabpdu1.flag[3] = 0;
}
...
```

Listing 14: Verschlüsselung der Portart im Flag-Feld

Nachdem nun die Proposal-BPDUs verschickt wurden, gehen die Ports in einen Wartemodus, den sie erst verlassen, sobald die entsprechende Agreement-BPDU eingetroffen ist, oder spätestens nach der siebenfachen Hello-Time.
Sobald eine Proposal-BPDU ankommt, geht der Port in den Learning-Status und merkt sich die Root-ID, die Root-Pathcosts, die Bridge-ID und die Portart des Sender-Ports. Dann wird eine Kopie der Nachricht erstellt und anstatt dem Proposal-Bit das Agreement-Bit gesetzt und an den Sender zurückgeschickt. Anschließend beginnt die Auswertung der empfangenen Informationen, um den Root-Switch zu bestimmen und vorhandene Schleifen aufzubrechen, indem bestimmte Ports blockiert werden.
Als Erstes wird kontrolliert, ob in der Nachricht ein besserer Root-Switch genannt wird, als der bisher bekannte. Dieser Test ist in Listing 15 zu sehen. Die Kriterien, nach denen entschieden wird, welcher Port der bessere ist, sind:

- Priorität
- ID
- Mac-Adresse

Der Switch vergleicht seine Werte in der angegebenen Reihenfolge mit dem empfangenen Wissen. Wenn in der BPDU das geprüfte Kriterium kleiner ist, übernimmt der Switch die Root-ID und die Root-Pathcosts. Sollte es gleich sein, wird das nächste getestet. Ansonsten behält der Switch sein Wissen. In dem Fall, dass ein neuer Root-Switch gefunden wurde, wird das Wissen gespeichert und der Empfänger-Port zum Root-Port gemacht. Im Then-Teil der ersten If-Bedingung ist dieser Prozess vollständig dargestellt. Beim Bestimmen des Root-Ports muss beachtet werden, ob es bereits einen gibt. Falls ja, wird dieser wieder zu einem Designated-Port umgewandelt. Die durch Punkte verkürzten Bereiche sind analog zu dem bereits beschriebenen und wurden aus Platzgründen weggelassen.

```
// BPDU kleinere Prioritaet
if (Integer.parseInt(part1[0]) < Integer.parseInt(part2[0])) {
   aktuellerSwitch.root_switch = false;
   aktuellerSwitch.label_switch.setForeground(Color.BLUE);
   root_id_aktuell = rstabpdu.root_id;
   root_pathcosts_aktuell = aport.root_pathcosts_zwischen;
   // Wenn noch kein Root-Port vorhanden
   if (test_root_port == false) {
      test_root_port = true;
      aport.art = Portart.ROOT_PORT;
      aport.label_port.setForeground(Color.GREEN);
      aport.status = Portstatus.FORWARDING;
   } else {
      for (int i = 0; i < ports.length; i++) {
         if (ports[i].id != aport.id
               && ports[i].art != Portart.EDGE_PORT) {
            ports[i].art = Portart.DESIGNATED_PORT;
            ports[i].label_port.setForeground(Color.BLUE);
         }
      }
      aport.art = Portart.ROOT_PORT;
      aport.label_port.setForeground(Color.GREEN);
      aport.status = Portstatus.FORWARDING;
   }
}
// BPDU gleiche Prioritaet
else if (Integer.parseInt(part1[0]) == Integer.parseInt(part2[0])) {
   // BPDU kleiner ID
   if (Integer.parseInt(part1[1]) < Integer.parseInt(part2[1])) {
      ...
   }
   // BPDU gleiche ID
   else if (Integer.parseInt(part1[1]) == Integer.parseInt(part2[1])) {
      // BPDU kleinere Mac-Adresse
      if (Long.parseLong(part1[2]) < Long.parseLong(part2[2])) {
         ...
      } ...
```

Listing 15: Neuen Root-Switch lernen

Falls bei der Überprüfung der oben erwähnten Kriterien herauskommt, dass alle Werte gleich sind, bedeutet dies, dass der Empfänger-Switch bereits den Root-Switch kennt. Dann muss geschaut werden, ob in der BPDU vielleicht ein kürzerer Pfad zum Root-Switch enthalten ist. Dies ist in Listing 16 dargestellt. Dazu werden die folgenden Kriterien verwendet:

- Root-Pathcosts
- Priorität
- ID

Wie im Listing zu erkennen, werden zuerst die empfangenen Root-Pathcosts mit denen vom Switch verglichen. Wenn die von der BPDU kleiner sind, wird der Empfangsport zum Root-Port und alle anderen außer den Edge-Ports zu Designated-Ports. Sollten die Root-Pathcosts gleich

sein, werden der Reihe nach die anderen beiden Werte überprüft, und zwar diesmal zwischen dem Empfangsport und dem bisherigen Root-Port und nicht zwischen Empfangsport und BPDU. Spätesten beim Test der ID kann eine eindeutige Entscheidung getroffen werden.

```
// BPDU gleiche Mac-Adresse
else if (Long.parseLong(part1[2]) == Long.parseLong(part2[2])) {
   // BPDU kleinere Root-Pathcosts
   if (rstabpdu.root_pathcosts + aport.wire.path_costs < root_pathcosts_aktuell) {
      // Setze alle anderen Ports auf Designated
      for (int i = 0; i < ports.length; i++) {
         if (ports[i].id != aport.id && ports[i].art != Portart.EDGE_PORT) {
            ports[i].art = Portart.DESIGNATED_PORT;
            ports[i].label_port.setForeground(Color.BLUE);
         }
      }
      aport.art = Portart.ROOT_PORT;
      aport.label_port.setForeground(Color.GREEN);
      aport.status = Portstatus.FORWARDING;
      root_pathcosts_aktuell = aport.root_pathcosts_zwischen;
   }
   // BPDU gleiche Root-Pathcosts
   else if (rstabpdu.root_pathcosts + aport.wire.path_costs == root_pathcosts_aktuell) {
      ...
      // Empfangsport kleiner Prioriaet als bisherige Root-Port
      if (aport.priority < bport.priority) {
         ...
      }
      // Empfangsport gleiche Prioriaet als bisherige Root-Port
      else if (aport.priority == bport.priority) {
         // Empfangsport kleiner ID als bisherige Root-Port
         if (aport.id < bport.id) {
            ...
         }
      }
   }
}
```

Listing 16: Besserer Pfad zu bekanntem Root-Switch

Nachdem nun der beste Root-Switch und die Root-Ports gefunden wurden, werden die anderen verbleibenden Ports außer den Edge-Ports betrachtet. Zuerst werden die Backup-Ports vom Root-Port bestimmt, da dies am einfachsten umzusetzen ist und es keine Schwierigkeiten gibt sie herauszufinden. Im Listing 17 ist dieser Aspekt zu erkennen.

```
// Gegenueberliegende Port und Empfangsport duerfen kein Root-Port sein
if (!aport.portart_gegenueber.equals("10") && aport.art != Portart.ROOT_PORT) {
   // Pruefe, ob bereits ein Root-Port bestimmt wurde und ob
   // Empfangsport selbe Bridge-ID hat wie Root-Port
   if (bridge_id_root != null && aport.bridge_id_zwischen.equals(bridge_id_root)) {
      String[] part3 = rstabpdu.bridge_id.split(Pattern.quote("."));
      // Wenn Root-Pfadkosten des aktuellen Switches groesser sind als gegenueberliegende
      if (root_pathcosts_aktuell > rstabpdu.root_pathcosts) {
         aport.art = Portart.BACKUP_PORT;
         aport.status = Portstatus.DISCARDING;
         aport.label_port.setForeground(Color.ORANGE);
      }
      // Wenn aktueller Switch gleiche Root-Pfadkosten hat wie gegenueberliegende
      else if (root_pathcosts_aktuell == rstabpdu.root_pathcosts) {
         // Wenn aktueller Switch schlechtere Prioritaet hat als gegenueberliegende,
         // dann wird aktueller Port blockiert
         if (aktuellerSwitch.priority > Integer.parseInt(part3[0])) {
            ...
         }
         // Wenn aktueller Switch gleiche Prioritaet hat wie gegenueberliegende
         else if (aktuellerSwitch.priority == Integer.parseInt(part3[0])) {
            // Wenn aktueller Switch schlechtere ID hat als gegenueberliegende,
            // dann wird aktueller Port blockiert
            if (aktuellerSwitch.id > Integer.parseInt(part3[1])) {
               ...
            }
            // Wenn aktueller Switch gleiche ID hat wie gegenueberliegende
            else if (aktuellerSwitch.id == Integer.parseInt(part3[1])) {
               // Wenn aktueller Switch schlechtere Mac-Adresse hat als gegenueberliegende,
               // dann wird aktueller Port blockiert
               if (Integer.parseInt(aktuellerSwitch.mac_address) > Integer.parseInt(part3[2])) {
                  ...
               }}}}}}
```

Listing 17: Backup-Port zu Root-Port bestimmen

Die erste Bedingung dient nur zur Sicherstellung, dass der Empfangsport und der gegenüberliegende Port auch kein Root-Port ist. Dies ist wichtig, damit ein bereits erkannter Root-Port nicht zu einem Backup-Port umgewandelt werden kann und dass der Port gegenüber einem Root-Port ein Designated-Port bleibt. Danach findet die entscheidende Prüfung statt, und zwar, ob der Empfangsport zum selben Switch führt wie der Root-Port. Wenn dem so ist, wird mittels der restlichen If-Bedingungen nur noch kontrolliert, welches Ende der Leitung zum Backup-Port wird. Die dazu notwendigen Kriterien wurden bereits im Unterunterkapitel 2.3.3 aufgeführt. Sobald der richtige Port gefunden wurde, wird er zum Backup-Port und wird in den Discarding-Status versetzt.

Wenn alle infrage kommenden Leitungen blockiert wurden, werden die Alternate-Ports ermittelt. Als Erstes werden alle Ports gesucht, die zum selben Switch führen wie der Empfangsport. Dies ist in Listing 18 dargestellt.

```
ArrayList<Port> ports1 = new ArrayList<Port>();
int test_root = 0;
// Gegenueberliegende Port und Empfangsport duerfen kein Root-Port sein
if (!aport.portart_gegenueber.equals("10") && aport.art != Portart.ROOT_PORT) {
   // Nur wenn Empfangsport ueber das aktuelle Wissen von Root-Switch verfuegt,
   // kann Alternate-Port bestimmt werden
   if (root_id_aktuell != null && aport.root_id_zwischen.equals(root_id_aktuell)) {
      // Pruefe, ob bereits ein Root-Port bestimmt wurde und ob Empfangsport nicht
      // dieselbe Bridge-ID hat wie Root-Port
      if (bridge_id_root != null && !aport.bridge_id_zwischen.equals(bridge_id_root)) {
         // Suche alle anderen Ports, dieselbe Bridge-ID haben wie Empfangsport
         for (int i = 0; i < ports.length; i++) {
            // Ueberspringe alle Edge-Ports und alle deaktivierten Ports
            if (ports[i].art != Portart.EDGE_PORT
                  && ports[i].test_deaktivierter_port == false) {
               // Wenn Port schonmal eine BPDU mit dem aktuellen Wissen
               // ueber den Root-Switch empfangen hat
               if (ports[i].bridge_id_zwischen != null
                        && ports[i].root_id_zwischen.equals(root_id_aktuell)) {
                  // Wenn Port selbe Bridge-ID hat wie Empfangsport
                  if (aport.bridge_id_zwischen.equals(ports[i].bridge_id_zwischen)) {
                     // Es darf keinen Port mit derselben Bridge-ID wie Empfangsport geben,
                     // der Root-Port ist, da dann all diese Leitungen Backup-Leitungen
                     // des gegenueberliegenden Switches sind
                     if (!ports[i].portart_gegenueber.equals("10")) {
                        ports1.add(ports[i]);
                        test_root = 1;
                     } else {
                        ports1.clear();
                        test_root = 2;
                        break;
                     }}
               }
               // Wenn es einen Port gibt, der noch keine BPDU empfangen hat
               // oder nicht ueber das aktuelle Wissen ueber den Root-Switch verfuegt,
               // kann Alternate-Port noch nicht bestimmt werden
               else {
                  ports1.clear();
                  test_root = 0;
                  break;
               }}}}}}
```

Listing 18: Suche alle Ports mit derselben Bridge-ID wie Empfangsport

Die erste If-Bedingung ist dieselbe wie im vorherigen Listing. Als Zweites wird getestet, ob der Empfangsport bereits eine BPDU empfangen hat, in der das Wissen über den aktuellen Root-Switch enthalten war. Sollte der Test negativ ausfallen, kann nicht herausgefunden werden, ob der Port überhaupt zum Root-Switch führt und kommt deshalb auch nicht als Alternate-Port infrage. In der dritten Bedingung findet der entscheidende Test statt, und zwar, ob der Empfangsport zu einem anderen Switch führt wie der Root-Port. Damit ist sichergestellt, dass der aktuelle Port kein Backup-Port ist. In der For-Schleife werden dann alle Ports gesucht, die zum selben Switch führen

wie der Empfangsport. Dabei werden alle Edge-Ports, deaktivierten Ports und Ports, die noch keine Nachricht vom aktuellen Root-Switch empfangen haben, übersprungen. In der innersten If-Bedingung wird geschaut, ob einer der gefundenen Ports einen gegenüberliegenden Root-Port hat. Sollte dies nicht der Fall sein, wird der Port in die ArrayList ports1 eingefügt und die Variable test_root auf 1 gesetzt. Diese Varibale dient zum Feststellen, ob ein Root-Port gefunden wurde oder ob ein Port noch nicht über das aktuelle Wissen über den Root-Switch verfügt. Bei einem gefunden Root-Switch wird die Variable auf 2 und in der verbleibenden Situation auf 0 gesetzt. Wenn ein Root-Port entdeckt wurde, darf keiner der gefundenen Ports ein Alternate-Port werden, da diese Leitungen Backup-Leitungen sind. Dann wird die ArrayList ports1 gelöscht und die Schleife sofort verlassen. Auf diese Weise ist garantiert, dass sobald ein Root-Switch gefunden wurde, die Suche nach weiteren möglichen Alternate-Ports abgebrochen wird.

Falls kein gegenüberliegender Root-Port gefunden wurde, muss geprüft werden, ob der Empfangsport von allen gefundenen Ports der beste ist. Die Vorgehensweise ist in Listing 19 zu erkennen.

```
int beste_alternate_leitung = 0;
// Wenn kein Root-Port gefunden wurde
if (test_root == 1) {
   // Wenn es nur eine Leitung zum anderen Switch gibt, ist diese in
   // jedem Fall die beste Alternate-Leitung
   if (ports1.size() == 1) {
      beste_alternate_leitung = 1;
   }
   // Wenn es mehr als eine Leitung zum anderen Switch gibt, muss
   // zuerst einmal die beste Leitung bestimmt werden
   else if (ports1.size() > 1) {
      for (int i = 0; i < ports1.size(); i++) {
         // Ueberspringe Empfangsport
         if (aport.id != ports1.get(i).id) {
            // Wenn Empfangsport niedrigere Root-Pfadkosten hat
            if (aport.root_pathcosts_zwischen < ports1.get(i).root_pathcosts_zwischen) {
               beste_alternate_leitung = 1;
            }
            // Wenn Empfangsport gleiche Root-Pfadkosten hat
            else if (aport.root_pathcosts_zwischen == ports1.get(i).root_pathcosts_zwischen) {
               ...
                  if (aport.id < ports1.get(i).id) {
                     beste_alternate_leitung = 1;
                  }
                  // Wenn Empfangsport groessere ID hat
                  else if (aport.id > ports1.get(i).id) {
                     beste_alternate_leitung = 2;
                  }
               ...
```

Listing 19: Test auf Empfangsport

Wenn es in der ArrayList nur einen Port gibt, ist dieser in jedem Fall der Empfangsport und somit auch der Alternate-Port. Dann wird die Variable beste_alternate_leitung auf 1 gesetzt. Sollten sich mehrere Ports in der Liste befinden, muss der beste bestimmt werden.

Dazu werden die folgenden Kriterien verwendet:

- Root-Pathcosts
- Priorität
- ID

Der Empfangsport muss diese Werte der Reihe nach mit allen Ports aus der ArrayList vergleichen. Wenn er die kleineren Werte hat, wird die Variable auf 1, wenn er die größeren hat auf 2 gesetzt. Der aktuelle Port wird dabei nicht mit sich selbst verglichen. Sollte der Empfangsport der beste Port sein, wird geprüft, welches Ende der Leitung zum Alternate-Port wird. Wenn er nicht der beste ist, wird er zum Backup-Port. Auch hierfür muss vorher noch das richtige Ende der Leitung ermittelt werden.
Falls in Listing 18 ein gegenüberliegender Root-Port gefunden wurde, wird entweder der Empfangsport oder sein gegenüberliegender Port zum Backup-Port.

Irgendwann empfängt ein Port die erste Agreement-BPDU, wodurch er sofort in den Forwarding-Status gesetzt wird. Von nun an verschickt der Switch über ihn entsprechend der Hello-Time normale BPDUs. Wenn aber innerhalb der siebenfachen Hello-Time keine Agreement-BPDU eintrifft, wird der Port mittels der aus dem ursprünglichen Spanning Tree Algorithmus bekannten Forward-Delay-Timern in den Forwarding-Status versetzt. Dieser Vorgang ist in Listing 20 zu erkennen. Mittels der For-Schleife wird über alle Ports des Switches iteriert und dann getestet, ob sie überhaupt auf eine Agreement-BPDU warten. Wenn dem so ist, wird geprüft, ob die siebenfache Hello-Time bereits abgelaufen ist. Sollte auch dies zutreffen, muss festgestellt werden, ob der aktuelle Port ein Root-Port oder ein Designated-Port ist. Denn nur diese beiden Portarten dürfen überhaupt in den Forwarding-Status gesetzt werden. Anschließend wird überprüft, in welchem Status sich der Port bereits befindet. Wenn er sich im Discarding-Status befindet, muss er erst einmal in den Learning-Status wechseln, bevor er in den Forwarding-Status gelangen kann. Befindet er sich im Learining-Status erübrigt sich der Zwischenschritt. In beiden Fällen legt sich dann der Switch-Thread entsprechend des Forward-Delay-Timers schlafen. Nach dem Aufwachen wird geprüft, ob die im Timer definierte Zeit abgelaufen ist und bei erfolgreichem Test wechselt der Port in den nachfolgenden Status, also entweder in den Learning- oder direkt in den Forwarding-Status. Falls es notwendig ist, wird dies nochmal wiederholt, damit auch alle entsprechenden Ports im Forwarding-Status sind.

```
for (int i = 0; i < ports.length; i++) {
   // Wenn Port auf Agreement-BPDU wartet
   if (ports[i].test_prosposal == true) {
      // Wenn Agreement-BPDU nicht rechtzeitig eingetroffen ist
      if (ports[i].last_prosposal + aktuellerSwitch.hello_time * wartezeit * 7
            < System.currentTimeMillis()) {
         // Wenn Port Root-Port oder Designatet-Port ist
         if (ports[i].art == Portart.ROOT_PORT || ports[i].art == Portart.DESIGNATED_PORT) {
            ports[i].test_prosposal = false;
            // Wenn sich Port im Discarding-Status befindet
            if (ports[i].status == Portstatus.DISCARDING) {
               // Timer starten
               ports[i].start_forward_delay = System.currentTimeMillis();
               ...
               // Swicht legt sich entsprechend des Forward-Delay-Timer schlafen
               try {
                  Switch.sleep(aktuellerSwitch.forward_delay * wartezeit);
               } catch (Exception e) {
                  aktuellerSwitch.interrupt();
                  return;
               }
               ...
               // Sobald Timer abgelaufen ist, gehe in den Learning-Status
               if (System.currentTimeMillis() >= ports[i].start_forward_delay
                     + aktuellerSwitch.forward_delay  * wartezeit) {
                  ports[i].status = Portstatus.LEARNING;
                  ...
```

Listing 20: Ausbleiben einer Agreement-BPDU

Nachdem der Aufbau des Spannbaums mithilfe des Rapid Spanning Tree Algorithmus erklärt wurde, wird im folgenden Unterkapitel auf die Rekonfiguration des Spannbaums eingegangen. [Wik11, Ava10, Bad05, Cis06, eTu12, Jan10b]

7.2 Rekonfiguration des Spannbaums beim RSTA

Das zweite Unterkapitel befasst sich mit der Rekonfiguration des Spannbaums. Diese Funktionalität des Rapid Spanning Tree Algorithmus wird gebraucht, falls die Leitung vom Root-Port eines Switches ausfällt. In so einem Fall hat der betroffene Switch keine aktive Verbindung mehr zum Root-Switch. Um weiterhin Nutznachrichten von einem Host des Netzwerks zu einem anderen verschicken zu können, muss der Spannbaum entsprechend angepasst werden.

7.2.1 Arbeitsweise der Rekonfiguration beim RSTA

Die Ausgangssituation für die Rekonfiguration ist ein vollständig aufgebauter Spannbaum, der in Abbildung 46 zu sehen ist. Zum besseren Verständnis wird hier dasselbe Netzwerk verwendet wie in Unterkapitel 7.1.

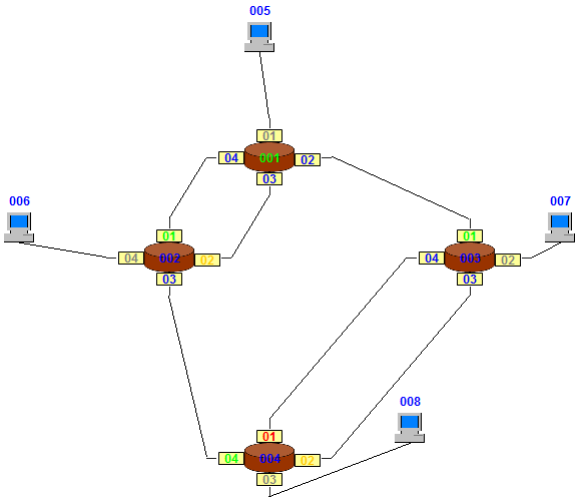


Abbildung 46: Vollständig aufgebauter Spannbaum

Es besteht aus vier miteinander verbundenen Switches, von denen der Switch 001 der Root-Switch ist. Alle normalen Switches haben genau einen grünen Root-Port. Des Weiteren gibt es vier graue Edge-, zwei orangene Backup- und einen roten Alternate-Port. Die Bedeutung dieser Portarten wurde bereits im Unterunterkapitel 2.3.3 erklärt. Das hier abgebildete Netzwerk ist stabil, was bedeutet, dass es sich ohne Beeinflussung durch den Nutzer nicht mehr verändern wird.

Dem Anwender steht es nun frei, jede beliebige Leitung ausfallen zu lassen. Jedoch um eine Rekonfiguration des Spannbaums auszulösen, muss die Leitung eines Root-Ports ausfallen, da nur dort ein Austausch von BPDUs in beiden Richtungen stattfindet. Wenn eine bereits blockierte Leitung oder eine Edge-Leitung ausfällt, spielt es für den Rapid Spanning Tree Algorithmus und somit für den Spannbaum keine Rolle.

Um alle relevanten Aspekte der Rekonfiguration zu erklären, wird erstmal die Root-Leitung zwischen Switch 002 und 001 ausfallen gelassen. Dies ist in Abbildung 47 dargestellt.

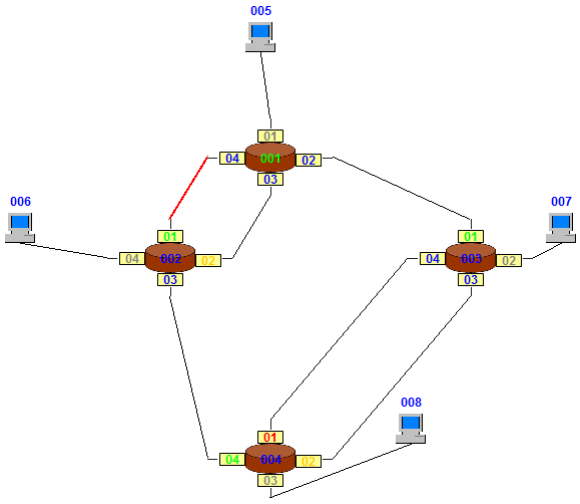
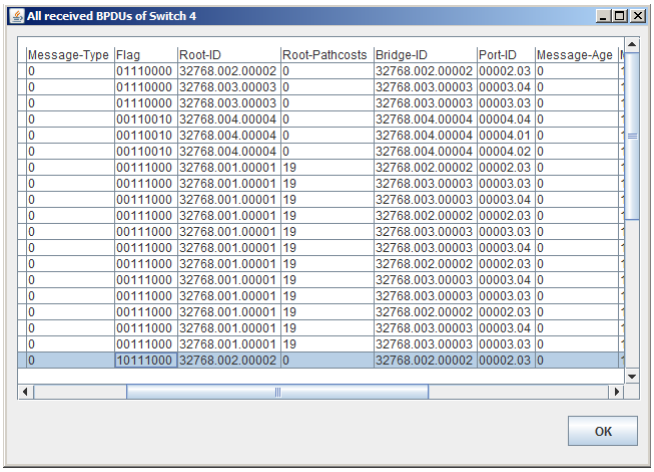


Abbildung 47: Ausgefallene Root-Leitung von Switch 002

An der roten Einfärbung, kann die deaktivierte Leitung erkannt werden. Nun findet keinerlei Nachrichtenverkehr mehr über diese Verbindung statt und Switch 002 wird nach Ablauf des Timeout-Intervalls ein Timeout bekommen. Auch hier tritt das schon mehrfach angesprochene Problem auf, dass der Simulator langsamer als ein reales Netzwerk ist. Deshalb muss das Intervall im vorliegenden Beispiel anstatt auf die dreifache auf die siebenfache Hello-Time verlängert werden, was aber im Vergleich zum ursprünglichen Spanning Tree Protokoll immer noch eine merkliche Beschleunigung darstellt.

Sobald Switch 002 ein Timeout bekommt, vergisst er die Root-ID und die Root-Pathcosts vom Root-Switch. Dann setzt er all seine Ports bis auf den Edge-Port in die Ausgangssituation zurück. Das bedeutet, dass sie alle zu Designated-Ports werden und den Discarding-Status erhalten. Dann geht er davon aus, dass er selbst der Root-Switch ist. Anschließend verschickt er mittels Broadcast an all seine Nachbarn eine Topology-Change-BPDU. Dies ist eine normale BPDU, bei der nur das Topology-Change-Bit im Flag-Feld gesetzt wird. Zum Schluss geht er in einen Wartemodus und startet den „TC While Timer“, der der zweifachen Hello-Time entspricht. Dies ist notwendig, damit das restliche Netzwerk Zeit hat, die TC-BPDU zu empfangen und entsprechend zu reagieren, bevor der aktuelle Switch beginnt wieder Proposal-BPDUs zu versenden oder auszuwerten. Während dieser Zeit macht der Switch gar nichts. Sobald der Timer abgelaufen ist, verlässt er selbstständig den Wartemodus und verschickt über alle Ports, die sich in der Ausgangssituation befinden Proposal-BPDUs. Nun startet der normale Prozess zum Aufbau des Spannbaums, der bereits im Unterkapitel 7.1 beschrieben wurde.

Solch eine Topology-Change-BPDU ist in Abbildung 48 zu erkennen.



Message-Type	Flag	Root-ID	Root-Pathcosts	Bridge-ID	Port-ID	Message-Age
0	01110000	32768.002.00002	0	32768.002.00002	00002.03	0
0	01110000	32768.003.00003	0	32768.003.00003	00003.04	0
0	01110000	32768.003.00003	0	32768.003.00003	00003.03	0
0	00110010	32768.004.00004	0	32768.004.00004	00004.04	0
0	00110010	32768.004.00004	0	32768.004.00004	00004.01	0
0	00110010	32768.004.00004	0	32768.004.00004	00004.02	0
0	00111000	32768.001.00001	19	32768.002.00002	00002.03	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.03	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.04	0
0	00111000	32768.001.00001	19	32768.002.00002	00002.03	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.03	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.04	0
0	00111000	32768.001.00001	19	32768.002.00002	00002.03	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.04	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.03	0
0	00111000	32768.001.00001	19	32768.002.00002	00002.03	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.04	0
0	00111000	32768.001.00001	19	32768.003.00003	00003.03	0
0	10111000	32768.002.00002	0	32768.002.00002	00002.03	0

Abbildung 48: Eine von Switch 004 empfangene Topology-Change-BPDU

Auch diese Abbildung zeigt nur nur den relevanten Teil der Nachricht. In der blau markierten Zeile ist die Topology-Change-BPDU an dem gesetzten ersten Bit des Flag-Felds zu erkennen. Wie in den Feldern Root-ID und Root-Pathcosts zu sehen, geht Switch 004 nun davon aus, dass er selbst der Root-Switch ist.

Sobald ein Switch solch eine Topology-Change-BPDU empfängt, vergisst auch er die Root-ID und die Root-Pathcosts. Er geht nun davon aus, dass er selbst der Root-Switch ist und verschickt über alle Root- und Designated-Ports eine eigene Topology-Change-BPDU. Erst danach setzt er alle seine Ports bis auf den Edge-Port in die Ausgangssituation zurück und startet den „TC While Timer“. Er verfährt von nun an genauso, wie vorher beschrieben. Diese Situation ist in Abbildung 49 dargestellt.

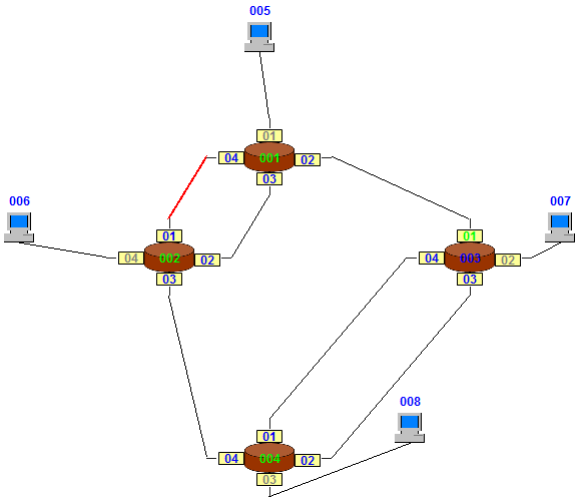


Abbildung 49: Switch 002 und 004 warten auf das Ablaufen des TC While Timers

An der grünen Beschriftung von Switch 002 und Switch 004 ist zu sehen, dass beide Switches davon ausgehen, dass sie selbst der Root-Switch sind. In der abgebildeten Situation haben beide den „TC While Timer“ gestartet und warten darauf, dass dieser abläuft. Im Gegensatz zum Spanning Tree Algorithmus IEEE 802.1D wird im vorliegenden Beispiel nicht der gesamte Spannbaum neu aufgebaut. Switch 003 ist hiervon nicht betroffen und hat auch keine Topology-Change-BPDU empfangen, da die Ports 01 und 02 des Switches 004 blockiert waren und deshalb darüber keine BPDU verschickt wurde.

Sobald der „TC While Timer“ abgelaufen ist, verlassen die Switches den Wartemodus und verschicken über alle Ports, die sich in der Ausgangssituation befinden Proposal-BPDUs. Auf diese Weise wird der neue Spannbaum aufgebaut, der in Abbildung 50 abgebildet ist.

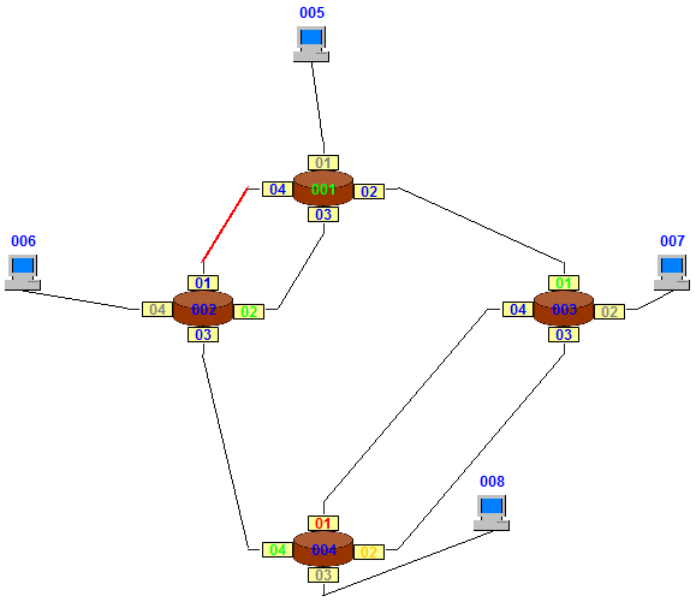


Abbildung 50: Der rekonfigurierte Spannbaum

Der neue Baum unterscheidet sich nur an zwei Stellen von dem aus Abbildung 46, und zwar einmal, dass der vorherige Root-Port nun deaktiviert und zum anderen, dass der frühere Backup-Port zum Root-Port wurde. Dieses Ergebnis ist auch so erwünscht, denn wenn die beste Leitung ausfällt, wird die zweitbeste genommen, welche in diesem Fall die Backup-Leitung ist. Für Switch 003 und 004 ergeben sich dadurch keinerlei Veränderungen.

Nun wird eine zweite Verbindung ausfallen gelassen, und zwar die Leitung zwischen Switch 001 und 003. Diesmal gibt es keine Backup-Leitung, die die ausgefallene ersetzten kann. In Abbildung 51 ist diese Situation zu sehen. Sobald bei Switch 003 das Timeout eintritt, vergisst er wieder sein gesamtes Wissen über den Root-Switch und geht davon aus, dass er es selbst ist. Dann verschickt er auch an seine Nachbarn Topology-Change-BPDUs, die, wie vorher schon beschrieben, reagieren. Da Switch 004 die Nachricht nun nicht mehr über seinen Root-Port empfängt, leitet er diese auch an Switch 002 weiter, was dazu führt, dass diesmal das gesamte Netzwerk von der Rekonfiguration betroffen ist.

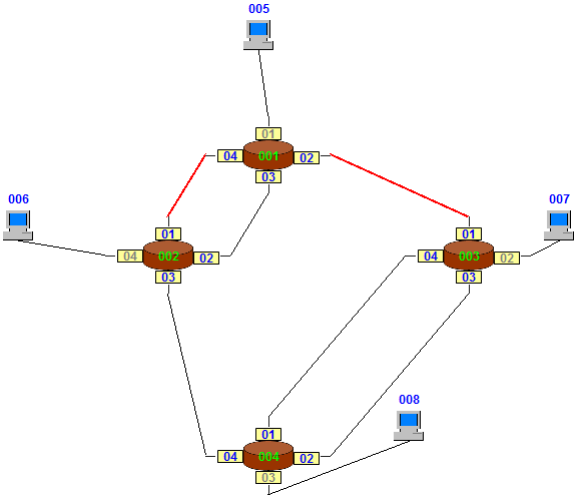


Abbildung 51: Leitung zwischen Switch 001 und 004 fällt auch noch aus

In Abbildung 52 ist der neue Spannbaum dargestellt. Da der Root-Port von Switch 003 ausgefallen ist, muss ein neuer bestimmt werden. Die einzige noch verbleibende Verbindung zum Root-Switch 001 ist die Root-Leitung von Switch 002. Also muss Switch 003 entweder Port 03 oder 04 zum Root-Port machen, da darüber Switch 001 noch erreichbar ist. Die Wahl fällt auf Port 03, weil dieser die kleinere ID hat. Port 04 muss nun zwangsläufig zum Backup-Port werden, da er auch zu Switch 004 führt. Alternate-Ports gibt es jetzt keine mehr, da es keinen alternativen Pfad zum Root-Switch gibt.

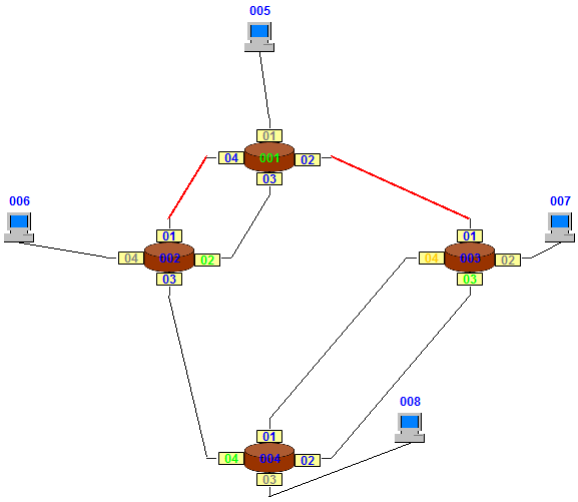


Abbildung 52: Der rekonfigurierte Spannbaum

Als nächstes wird jetzt die letzte noch verbliebene Leitung zum Root-Switch 001 ausfallen gelassen. Dies ist in Abbildung 53 zu sehen. Sobald Switch 002 ein Timeout bekommt, vergisst er wieder die Root-ID und die Root-Pathcosts und geht davon aus, dass er selbst der Root-Switch ist. Auch diesmal ist das gesamte Netzwerk von der Rekonfiguration betroffen und die Switches 002, 003 und 004 starten dementsprechend ihren TC While Timer. Da der bisherige Root-Switch nicht mehr erreichbar ist, muss ein neuer gefunden werden.

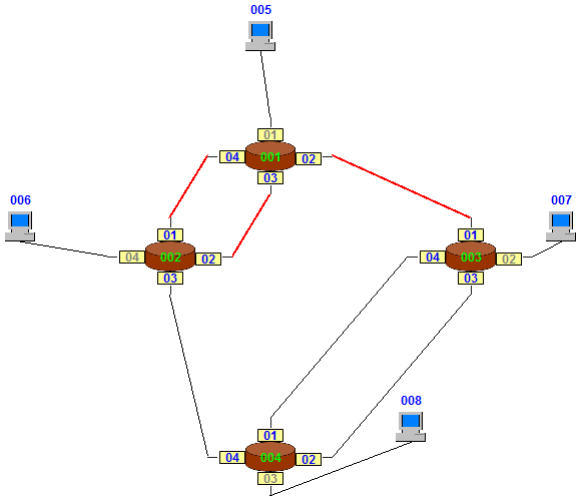


Abbildung 53: Ausfall der letzten Verbindung zum Root-Switch

Zum Schluss folgt nun noch Abbildung 54, die den fertigen Spannbaum mit dem neuen Root-Switch zeigt.

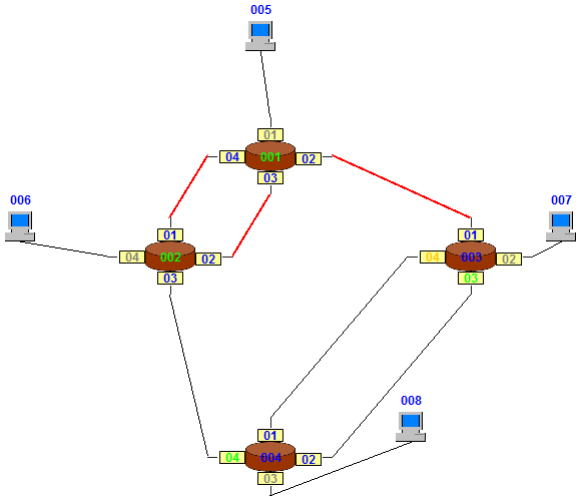


Abbildung 54: Der rekonfigurierte Spannbaum mit neuem Root-Switch

Da Switch 001 nicht mehr erreichbar ist, musste ein neuer Root-Switch bestimmt werden. Die Wahl fiel anhand der in Unterkapitel 7.1 beschriebenen Kriterien auf Switch 002, weil dieser die niedrigste ID hatte. Dementsprechend haben die beiden anderen Switches ihre Root-Ports bestimmt. Da es zwischen Switch 003 und 004 immer noch eine Schleife gibt, muss Port 04 des Switches 003 zum Backup-Port gemacht werden. Einen Alternate-Port gibt es auch hier nicht, weil es nur einen möglichen Pfad zum Root-Switch 002 gibt.

Alles in allem ist der große Unterschied zum ursprünglichen Spanning Tree Algorithmus der, dass die Switches, die eine Topology-Change-BPDU empfangen, zuerst eigene BPDUs verschicken und erst dann ihre Ports zurücksetzen. Dadurch wird nicht immer das gesamte Netzwerk rekonfiguriert, was zu einer Zeitersparnis führt. Auch müssen die Switches nicht mehr auf eine Antwort warten, sondern verlassen nach Ablauf des Timers selbstständig den Wartemodus. All dies führt zu der Beschleunigung bei der Rekonfiguration des Spannbaums. [Wik11, Ava10, Bad05, Cis06, eTu12, Jan10b]

7.2.2 Implementation der Rekonfiguration beim RSTA

In diesem Unterunterkapitel wird erklärt, wie die Rekonfiguration des Spannbaums beim Rapid Spanning Tree Algorithmus implementiert wurde. Der erste Schritt dabei ist das Feststellen eines aufgetretenen Timeouts. Anders als beim ursprünglichen Rapid Spanning Tree Protokoll gibt es keinen Timer für den gesamten Switch mehr, sondern jeder Port hat einen eigenen. Deshalb muss regelmäßig der Root-Port auf eine eventuelle Überschreitung des Timeout-Intervalls hin überprüft werden. Die anderen Ports müssen nicht getestet werden, da deren Leitung entweder zu einem Host führt oder blockiert wird. Die verbleibenden Designated-Ports, auf die diese beiden Kriterien nicht zutreffen, können beim Testen trotzdem weggelassen werden, da sie dann Root-Leitungen von benachbarten Switches sind. Also spielt das Überschreiten des Timeout-Intervalls nur beim Root-Port eine Rolle. Wie im vorherigen Unterunterkapitel beschrieben, liegt dieses beim vorliegenden Netzwerk bei der siebenfachen Hello-Time. Im Listing 21 ist die eben erwähnte Überprüfung zu sehen.

```
// Pruefung, ob bei einem Port ein Timeout eingetreten ist
Boolean timeout_eingetreten = false;
// Nur wenn Rekonfiguration aktiviert ist
if (GUI.checkbox_reconfiguration.isSelected() == true) {
   for (int i = 0; i < ports.length; i++) {
      // Nur wenn Port Root-Port ausfaellt, spielt es fuer aktuellen
      // Switch eine Rolle
      if (ports[i].art == Portart.ROOT_PORT) {
         // Wenn Port schonmal eine BPDU empfangen hat
         if (ports[i].last_received_BPDU != 0) {
            // Wenn Time-Out-Intervall eines Ports abgelaufen ist
            if (ports[i].last_received_BPDU + aktuellerSwitch.hello_time * wartezeit
                  * faktor_timeout < System.currentTimeMillis()) {
               timeout_eingetreten = true;
            }}}}}
```

Listing 21: Prüfung des Root-Ports auf Überschreitung des Timeout-Intervalls

Die erste If-Bedingung ermöglicht es dem Nutzer mithilfe der Checkbox „Acitvate reconfiguration of the spanning tree.“, die sich links oben im Hauptfenster des Simulators befindet, die Rekonfiguration des Spannbaums an- bzw. abzuschalten. Standardmäßig ist sie aktiviert. Mit der For-Schleife und der darin enthaltenen ersten If-Bedingung wird der Root-Port gesucht. Mit der vorletzten Bedingung wird sichergestellt, dass ein Timeout erst eintreten kann, wenn der Port schon mal eine BPDU empfangen hat. Das letzte und wichtigste Kriterium, befindet sich in der noch verbleibenden If-Bedingung. Dort wird überprüft, ob das Timeout-Intervall überschritten wurde. Sollte dem so sein, wird die Variable „timeout_eingetreten“ auf true gesetzt. Der Test setzt sich aus verschiedenen Faktoren zusammen und zwar von vorne nach hinten aus:

- dem Zeitpunkt, zu dem der Port die letzte BPDU empfangen hat
- der Hello-Time des Switches (hier 2 Sekunden)
- dem Faktor, um aus Sekunden die benötigten Millisekunden zu machen (hier 1000)
- dem Faktor, um das Timeout-Intervall festzulegen (hier 7)

Sollte beim Root-Port ein Timeout eingetreten sein, startet er den „TC While Timer“ des Root-Ports und setzt all seine Ports bis auf die Edge-Ports in die Ausgangssituation zurück. Dies bedeutet, dass sie zu Designated-Ports und in den Discarding-Status gesetzt werden. Ebenfalls vergessen sie all ihr gelerntes Wissen wie z.B. mit welchen Switches sie verbunden sind. Danach vergisst auch der Switch all sein Wissen über den Root-Port, also die Root-ID und die Root-Pathcosts und geht nun davon aus, dass er selbst der Root-Switch ist. Dieses Vorgehen ist in Listing 22 implementiert.

```
// Wenn ein Timeout eingetreten ist
if (timeout_eingetreten == true) {
   for (int i = 0; i < ports.length; i++) {
      // Setze alle Ports in Ausgangszustand
      ports[i].art = Portart.DESIGNATED_PORT;
      ports[i].status = Portstatus.DISCARDING;
      ports[i].label_port.setForeground(Color.BLUE);
      ports[i].bridge_id_zwischen = null;
      ports[i].tc_while_timer = System.currentTimeMillis();
      ports[i].test_timeout = true;
      ...
   }
   // Switch vergisst Wissen ueber Root-Switch und traegt seine eigenen
   // Werte ein
   ...
   root_id_aktuell = aktuellerSwitch.priority + "." + id_switch
         + "." + aktuellerSwitch.mac_address;
   aktuellerSwitch.label_root_id.setText(root_id_aktuell);
   root_pathcosts_aktuell = 0;
   aktuellerSwitch.label_root_pathcosts.setText(new Integer(
         root_pathcosts_aktuell).toString());
   // Aktueller Switch geht davon aus, dass er selbst Root-Switch ist
   aktuellerSwitch.root_switch = true;
   erster_durchlauf = true;
   aktuellerSwitch.inbox1.clear();
   topology_changed = true;
   ...
```

Listing 22: Timeout eingetreten

Danach versendet der Switch mittels Broadcast über alle Ports außer den Edge-Ports eine Topology-Change-BPDU. Dies ist eine ganz normale BPDU, bei der nur das erste Bit des Flag-Felds auf 1 gesetzt wird. In Listing 23 ist dies zu sehen.

```
// Es werden eine TC-BPDU ueber jeden Port rausgeschickt
for (int i = 0; i < ports.length; i++) {
   // Ueberspringe alle Edge-Ports und alle deaktivierten Ports
   if (ports[i].art != Portart.EDGE_PORT && ports[i].test_deaktivierter_port == false) {
      // Erzeugen und Verschicken einer TC-BPDU
      RSTABPDU rstabpdu1;
      ports[i].status = Portstatus.LEARNING;
      bpdu_art = 0;
      rstabpdu1 = createRSTABPDU(ports[i]);
      String id_port = new Integer(ports[i].id).toString();
      if (id_port.length() == 1) {
         id_port = "0" + id_port;
      }
      rstabpdu1.port_id = aktuellerSwitch.mac_address + "." + id_port;
      ports[i].sendRSTABPDU(rstabpdu1);
      last_send_time = System.currentTimeMillis();
      ports[i].status = Portstatus.FORWARDING;
   }
}
```

Listing 23: Verschicken einer Topology-Change-BPDU

Mittels der For-Schleife und der ersten If-Bedingung wird über alle Ports iteriert, aber alle deaktivierten und Edge-Ports dabei übersprungen. Dann wird eine neue BPDU erzeugt und über jeden in Frage kommenden Port herausgeschickt.
Sobald dies erledigt ist, wartet der Switch bis der TC While Timer des Ports abgelaufen ist. Was er danach macht, ist in Listing 24 zu erkennen.

```
// Wenn TC-While-Timer abgelaufen ist, geht Port aus Timeout-Situation raus
for (int i = 0; i < ports.length; i++) {
   if (ports[i].test_timeout == true && ports[i].tc_while_timer
      + aktuellerSwitch.hello_time * wartezeit * 2 < System.currentTimeMillis()) {
      ports[i].test_timeout = false;
      // Nach Timeout-Situation soll Timer neu starten
      ports[i].last_received_BPDU = System.currentTimeMillis();
      topology_changed = false;
      aktuellerSwitch.inbox1.clear();
   }
}
```

Listing 24: TC While Timer ist abgelaufen

Mit der For-Schleife und der If-Bedingung wird getestet, ob der TC While Timer des Ports abgelaufen ist. Da dieser Code vom Switch-Thread regelmäßig aufgerufen wird, reicht hier ein einmaliges Testen der Ports aus. Bevor geschaut wird, ob die notwendige Zeit verstrichen ist, wird getestet, ob sich der Port überhaupt in einer Timeout-Situation befindet. Denn nur dann soll er die folgenden Codezeilen ausführen. Die Prüfung ist ähnlich aufgebaut wie die in Listing 21 und wird deshalb hier nicht nochmals erklärt. Sollte der Timer abgelaufen sein, geht der

Switch aus der Timeout-Situation raus, startet den normalen Timeout-Timer neu und löscht die Empfangsliste inbox1. Denn alle Nachrichten, die während dem Wartemodus empfangen wurden, sollen nicht ausgewertete werden, da er in diesem Zeitraum schließlich nicht aktiv war.

Was jetzt noch fehlt, ist der Empfang einer Topology-Change-BPDU. Da hier aber kaum Neues geschieht, gibt es dazu kein Listing. Als Erstes wird die Empfangsliste inbox1 gelöscht, damit keine BPDUs mit veraltetem Wissen ausgewertet werden. Dann schickt der Switch über alle Root- und Designated-Ports jeweils eine Topology-Change-BPDU. Erst danach setzt er all seine Ports außer den Edge-Ports in den Ausgangszustand zurück und startet den TC While Timer. Was dies alles bedeutet, wurde bereits weiter oben erklärt. Die Reihenfolge, damit zuerst die Nachrichten verschickt und dann die Ports zurückgesetzt werden, hat den Vorteil, dass diese TC-BPDUs nicht im ganzen Netzwerk verbreitet werden und somit auch nicht immer der gesamte Spannbaum neu aufgebaut werden muss. Als nächstes vergisst der Switch sein Wissen und geht davon aus, dass er selbst der Root-Switch ist.

Damit ist die Implementation der Rekonfiguration des Spannbaums mittels des Rapid Spanning Tree Algorithmus erklärt. [Wik11, Ava10, Bad05, Cis06, eTu12, Jan10b]

8 Senden einer Nachricht in einem Spannbaum

In den Kapiteln 2.2, 2.3 und 7 wurde bereits erklärt, wie die beiden implementierten Spanning Tree Algorithmen einen Spannbaum aufbauen. Dieser Baum ist notwendig, damit Host-Nachrichten die sogenannten Messages in Netzwerken mit Schleifen nicht im Kreis laufen. Die einzigen für die meisten Nutzer eines Netzwerks relevanten Nachrichten sind die Host-Nachrichten, die beispielsweise Nutzdaten von einem Host zu einem anderen übermitteln. Zur besseren Übersicht wird es in den zwei Unterkapitel die Funktionsweise und die Implementation des Versendens von Nachrichten getrennt voneinander erklärt.

8.1 Arbeitsweise des Nachrichtensendens im Spannbaum

Host-Nachrichten gab es auch schon in der Bachelorarbeit [Jan10b]. Jedoch wurden die damaligen Hosts noch als eigenständige Threads realisiert, die selbstständig in bestimmten Zeitintervallen Host-Nachrichten verschickten. Diese Realisierung hat sich jedoch für die Masterarbeit als ungünstig erwiesen, da zu viele Systemressourcen benötigt wurden. alle Switches, Hosts und Nachrichten wurden damals als eigenständige Threads realisiert, was dazu führte, dass der Simulator auf leistungsschwachen PCs nicht mehr richtig lief. Deshalb wurden nur noch die Switches als Threads realisiert. Ein weiterer Grund weshalb die Hosts nicht mehr selbstständig Nachrichten verschicken, ist der, damit Nutzer den genommenen Weg einer solchen Message leichter verfolgen können. Aus diesen beiden Gründen verschickt ein Host nur noch eine Nachricht, wenn der Anwender mittels des Kontextmenü-Punkts „Send Message to“ einer Host-Komponente explizit den Befehl dazu erteilt. Zu beachten ist hier noch, dass dies nur während einer laufenden Simulation möglich ist und das Netzwerk über mindestens zwei Hosts verfügen muss. Das Versenden einer Nachricht an den Sender-Host selbst wurde bewusst nicht realisiert. Um das Verschicken einer solchen Host-Nachricht und das Visualisieren des genommenen Wegs zu veranschaulichen, wird das Netzwerk in Abbildung 55 verwendet.

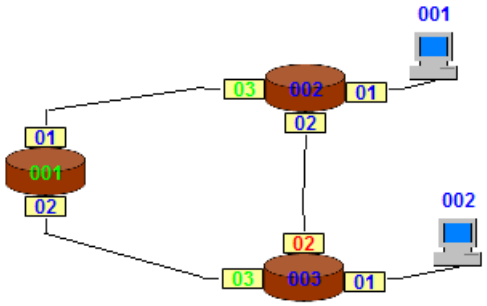


Abbildung 55: Ausgangssituation des Netzwerks zum Verschicken von Messages

Das abgebildete Netzwerk besteht aus nur drei Switches und zwei Hosts, damit der Ablauf dieses Prozesses einfacher nachvollzogen werden kann. Switch 001 ist wieder der Root-Switch und die normalen Switches 002 und 003 haben dementsprechend ihren Root-Port bestimmt. Um die vorhandene Schleife aufzubrechen, musste die Leitung zwischen den beiden Switches 002 und 003 blockiert werden. Deshalb wurde der Port 02 des Switches 003 zum Blocking-Port, was auch

an der roten Einfärbung der Schrift zu erkennen ist. Der Spannbaum wurde beim vorliegenden Netzwerk mithilfe des Spanning Tree Algorithmus IEEE 802.1D aufgebaut.

Wenn der Nutzer nun eine Message von Host 001 an Host 002 schicken möchte, kann er dies mittels des weiter oben erwähnten Kontextmenü-Punkts tun. Durch seine Betätigung wird eine Message erzeugt und diese an die an den Host angeschlossene Wire übergeben, die dann sofort grün eingefärbt wird. Diese Situation ist in Abbildung 56 dargestellt.

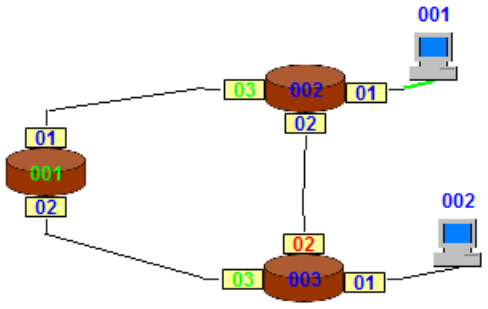


Abbildung 56: Host 001 übergibt Message an Wire

Nachdem Switch 002 diese Message empfängt, prüft er, ob der Empfangsport ein Root- oder ein Designated-Port ist und ob dieser sich im Forwarding-Status befindet. Denn nur in diesen Fällen, darf ein Switch eine Message auswerten und weiterleiten. Da Port 01 die Kriterien erfüllt, wertet der Switch die Nachricht aus. Der erste Schritt dabei ist das Lernen des Sender-Hosts. Da Port 01 ein Designated-Port ist, weiß Switch 002, dass die Nachricht entweder direkt von einem Host oder zumindest von einem Kind-Switch im Spannbaum stammt. Von einem Geschwister-Switch kann keine Message kommen, da der gegenüberliegende Port ein Blocking-Port ist und über diesen keine Nachrichten empfangen oder gesendet werden. Diese ganze Überprüfung ist wichtig, weil bei einem Spannbaum die gesamte Kommunikation nur über den Root-Switch verläuft. Das bedeutet, dass jede Host-Nachricht zunächst an den Wurzelknoten gesendet werden muss, um sie dann von dort aus zum Ziel-Host zu schicken.

Da Port 01 ein Designated-Port ist, wird nun überprüft, ob es zum Sender-Host bereits einen Eintrag in der SAT-MAC-Table gibt. Sollte dies nicht der Fall sein, wird er ergänzt. Solch ein Eintrag ordnet den Sender-Host dem Empfangsport zu, sodass der Switch von nun an weiß, über welchen Port dieser Host zu erreichen ist. Nach dem Lernen muss im zweiten Schritt die Nachricht weitergeschickt werden. Da die Message nicht über den Root-Port empfangen wurde, versendet Switch 002 sie über ihn. Dies ist in Abbildung 57 zu sehen.

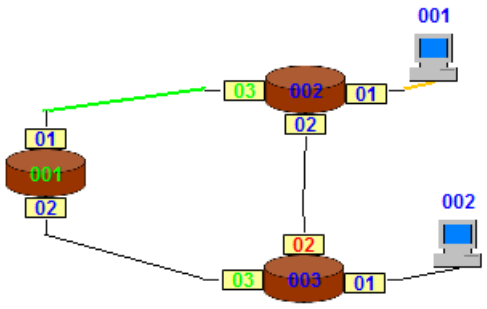


Abbildung 57: Switch 002 schickt Message an Root-Switch

Sobald Switch 002 die Message an die nächste Wire übergibt, wird diese grün und die mittlerweile verlassene orange eingefärbt. Der Root-Switch füllt seine SAT-MAC-Table auf dieselbe Art und Weise wie Switch 002. Da er den Ziel-Host nicht kennt, verschickt er mittels Broadcast über all seine Ports die empfangene Message, was in Abbildung 58 visualisiert ist.

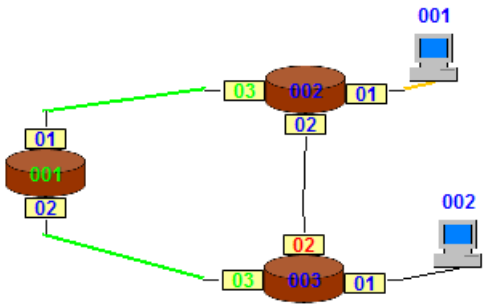


Abbildung 58: Root-Switch verschickt Message mittels Broadcast

Switch 002 und 003 empfangen beide die Message über ihren jeweiligen Root-Port. Nun müssen sie prüfen, ob es bereits einen Eintrag zum Sender-Host in ihrer SAT-MAC-Table gibt. Falls noch keiner vorhanden ist, wird er erstellt und der Table hinzugefügt. Diese Prüfung ist wichtig, weil ansonsten Switch 002 den Eintrag in der SAT-MAC-Table überschreiben würde, da die Nachricht zum zweiten Mal empfangen wurde. Jede Message wird zuerst immer an den Root-Switch geschickt und kann dann unter Umständen wegen einem ausgeführten Broadcast über den Root-Port erneut empfangen werden. Deshalb darf nur beim ersten Eintreffen der Message ein Eintrag erstellt werden.

Zum Weiterleiten schauen die Switches in ihren SAT-MAC-Tables nach, ob dort ein Eintrag zum Ziel-Host vorhanden ist. Da dies nicht der Fall ist, versenden beide Switches die Message

mittels Broadcast. Zu beachten ist, dass die Nachrichten weder über die geblockte Ports noch über die Empfangsports verschickt werden. Die Abbildung 59 zeigt den aktuellen Zustand nach dem Broadcast.

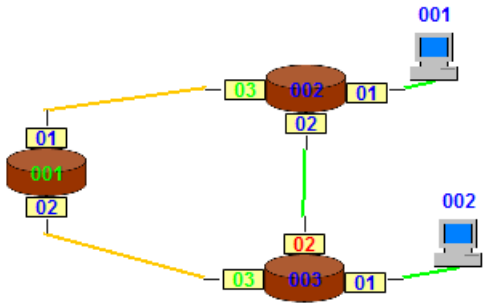


Abbildung 59: Switch 002 und 003 verschicken Message mittels Broadcast

Die beiden Leitungen des Root-Switches werden nun orange eingefärbt, da die Messages diese Wires bereits verlassen haben. Wie in Abbildung 59 zu sehen, haben letztendlich beide Hosts die Message empfangen. Die dritte grün eingefärbte Leitung ist eine blockierte Wire. Da Port 02 des Switches 002 ein Designated-Port ist, wurde beim Broadcast auch eine Message über ihn verschickt. Aber weil Port 02 des Switches 003 ein Blocking-Port ist, wird die Nachricht nicht empfangen und die Wire orange eingefärbt, was den finalen Zustand des Netzwerks darstellt (Abbildung 60).

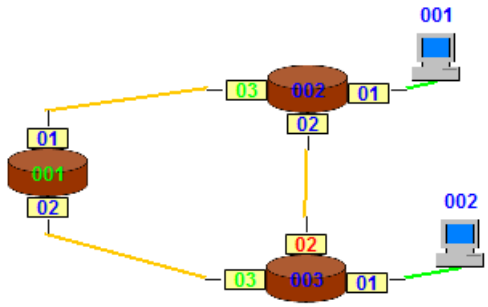


Abbildung 60: Finaler Zustand des Netzwerks

Wie eben beschrieben, haben beide Hosts die Message empfangen. Wenn nun der Nutzer eine Nachricht von Host 002 an Host 001 schicken möchte, funktioniert dies einfacher, da all Switches wissen, über welchen Port Host 001 zu erreichen ist. Um dieses Wissen zu veranschaulichen, folgt Abbildung 61, die die SAT-MAC-Table von Switch 003 nach dem Empfang der Message von Host 001 zeigt. Dort ist zu sehen, dass Switch 003 Host 001 über Port 03 erreichen kann. Über Port 02 gibt es zwar auch einen Weg, der aber blockiert ist.

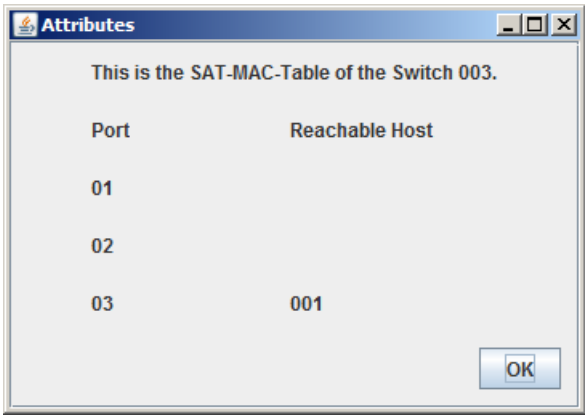


Abbildung 61: SAT-MAC-Table des Switches 003 nach Empfang der Message von Host 001

Die Ausgangssituation zum Verschicken einer Message von Host 002 zu Host 001 ist in Abbildung 60 dargestellt. Sobald der Nutzer den entsprechenden Kontextmenü-Punkt anklickt, werden alle grünen und orangenen Einfärbungen der Wires zurückgesetzt. Host 002 erzeugt eine Message und übergibt sie an die angeschlossene Wire. Switch 003 empfängt die Nachricht über den Designated-Port 01 und wertet sie aus. Dabei erstellt er den fehlenden Eintrag zu Host 002 in der SAT-MAC-Table und schickt die Host-Nachricht über den Root-Port zum Root-Switch. Dieser wertet die Message ebenfalls aus und verschickt sie diesmal nur über Port 01, da er weiß, dass darüber Host 001 zu erreichen ist. Dies ist in Abbildung 62 zu erkennen.

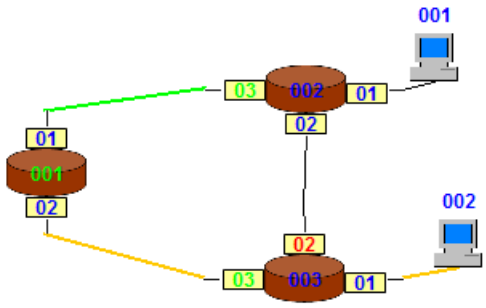


Abbildung 62: Root-Switch verschickt Message von Host 002 nur über Port 01

Auch Switch 002 verschickt die Nachricht nicht mehr mittels Broadcast, sondern gezielt über Port 01 an Host 001. In Abbildung 63 ist zu sehen, dass diesmal nur Host 001 die Message empfangen hat.

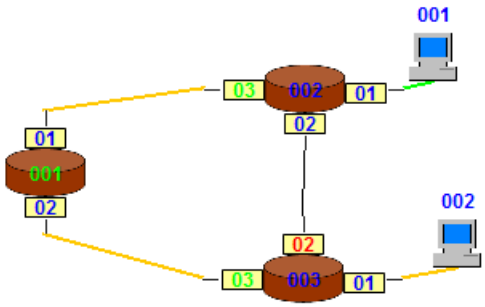


Abbildung 63: Finaler Zustand des Netzwerks

Nun wissen alle Switches, wo sich die beiden Hosts befinden, und können von nun an alle Messages gezielt verschicken. Das aktuelle Wissen der SAT-MAC-Table von Switch 003 ist in Abbildung 64 dargestellt. Dort ist zu erkennen, dass Host 001 über Port 03 und Host 002 über Port 01 zu erreichen ist.

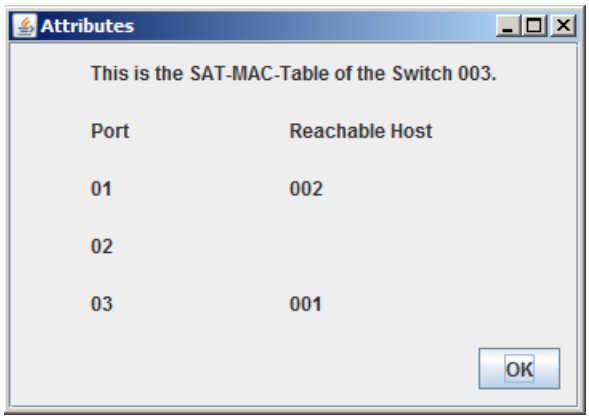


Abbildung 64: SAT-MAC-Table des Switches 003 nach Empfang der Message von Host 002

Bisher wurde erklärt, wie das Verschicken von Host-Nachrichten bei einem Spannbaum funktioniert, der mittels des Spanning Tree Algorithmus IEEE 802.1D aufgebaut wurde. Bei einem Spannbaum, der durch den Rapid Spanning Tree Algorithmus entstanden ist, bleibt zwar das Grundprinzip das gleiche, aber es gibt einige andere Portarten, deren Aufgabe bei dieser Prozedur noch geklärt werden müssen.
Der Blocking-Port wurde, wie in Kapitel 7 erwähnt, durch die beiden neuen Portarten Alternte- und Backup-Port ersetzt. Sie empfangen beide keine Messages und leiten sie auch nicht weiter. Die Root- und Designated-Ports werden weiterhin, wie weiter oben beschrieben, behandelt. Die letzte noch verbleibende Portart also die Edge-Ports, haben beim Verschicken der Host-Nachrichten dieselbe Funktion wie ein Designated-Port. [Jan10b]

8.2 Implementation des Nachrichtensendens im Spannbaum

Im vorherigen Unterkapitel wurde ausführlich erklärt, wie Messages von einem Host zu einem anderen verschickt und wie die benutzten Wires farblich markiert werden. Hier wird jetzt beschrieben, wie die wichtigsten Aspekte dieser Funktionalität implementiert wurden.

Um das Verschicken von Messages für den Nutzer möglichst einfach zu gestalten, wurden die Host-Komponenten mit dem Kontextmenü-Eintrag „Send Message to“ versehen. Dort werden alle Hosts aufsteigend nach ihrer Mac-Adresse sortiert aufgelistet. Der angeklickte Host fehlt in dieser Liste, damit der Nutzer keine Nachricht an ihn selbst schicken kann.

Sobald der Nutzer über den entsprechenden Menüpunkt den Befehl zum Verschicken einer Host-Nachricht gibt, wird ein Message-Objekt erzeugt. Dieses besteht nur aus den notwendigsten Informationen, die zum Verschicken einer Nachricht innerhalb eines Netzwerks notwendig sind. Diese sind:

- message_age
- max_age
- sender_mac_address
- ziel_mac_address

Die beiden Variablen message_age und max_age regeln das Alter einer Message. Damit ist sichergestellt, dass eine Nachricht nicht zu lange in einem Netzwerk unterwegs ist. Mit jedem Switch, den eine solche Nachricht passiert, wird das aktuelle Alter also die Variable message_age um 1 erhöht. Sobald sie das maximal erlaubte Alter von 16 erreicht hat, wird sie nicht mehr ausgewertet und weitergeleitet, sonder gelöscht. Da in diesem Kapitel die Messages in Spannbäumen laufen und die verwendeten Netzwerke eher klein sind, wird der Fall einer zu alten Nachricht nur sehr selten oder gar nicht eintreten. Im Kapitel 9 hingegen sind die beiden Variablen von essentieller Bedeutung, da dort Netzwerke mit nicht blockierten Schleifen vorliegen. In diesen würden die Messages ohne eine entsprechende Alterung endlos im Kreis laufen. Die beiden letzten Punkte also die Sender- und Ziel-Mac-Adresse sind eigentlich die bedeutendsten Werte, da nur durch sie ein Switch lernen kann, wo sich ein Host befindet und wohin er die Message weiterleiten soll.

Sobald eine Host-Nachricht erzeugt wurde, wird sie auf die an den Host angeschlossene Wire übergeben. Dazu müssen zunächst alle aktiven Wires wieder schwarz eingefärbt werden.

Wenn eine Wire ein Message-Objekt übergeben bekommt, wird sie grün dargestellt. Sobald die Nachricht von einem Switch empfangen wurde, wird die Leitung orange eingefärbt. Dies hat zur Folge, dass die Leitungen der Hosts, die eine Message empfangen haben, bis zum erneuten Verschicken einer Host-Nachricht grün bleiben. Dadurch kann der Anwender auch nach dem Ankommen einer Message beim Ziel-Host weiterhin erkennen, welcher Host sie empfangen hat.

Wie im vorherigen Unterkapitel beschrieben, erfolgen zuerst einmal eine ganze Reihe von Prüfungen, bevor sie weitergeleitet wird. Der erste Schritt stellt das Empfangen der Nachricht dar. Dieser ist im Listing 25 zu sehen. Der gezeigte Quellcode stammt aus der Klasse STA und re-

gelt somit das Empfangen von Messages beim Spanning Tree Algorithmus IEEE 802.1D. Auf das Zeigen des Quellcodes beim Rapid Spanning Tree Algorithmus wird wegen dem ähnlichen grundlegenden Prinzip verzichtet.

```
if (!inbox3.isEmpty()) {
   message1 = inbox3.peek();
   aktuellerSwitch.port_message_empfangen = (Port) message1.empfaenger;
   // Zeitverzoegerung beim Weiterleteiten einer Message, damit
   // Einfaerbung der Leitungen verfolgt werden kann
   if (message1.timestamp + 3000 < System.currentTimeMillis()) {
      // Wenn Port im Forwarding-Status ist und nicht geblockt wird
      if (aktuellerSwitch.port_message_empfangen.status == Portstatus.FORWARDING
            && aktuellerSwitch.port_message_empfangen.art != Portart.BLOCKING_PORT) {
         try {
            message1 = inbox3.take();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         aktuellerSwitch.port_message_empfangen = (Port) message1.empfaenger;
         empfange_message(aktuellerSwitch, ports);
      }
      // Wenn Port geblockt ist, loesche Message
      else if (aktuellerSwitch.port_message_empfangen.art == Portart.BLOCKING_PORT) {
         inbox3.remove();
         aktuellerSwitch.port_message_empfangen.wire.message_aktuell = false;
         aktuellerSwitch.port_message_empfangen.wire.message_besucht = true;
      }}}
```

Listing 25: Empfang einer Message

Die zweite If-Verzweigung des Listings 25 stellt eine Zeitverzögerung um drei Sekunden bei der Verarbeitung einer Message dar. Sie dient ausschließlich dem Zweck, damit der Nutzer den Prozess des Verschickens einer Nachricht besser verfolgen kann. Ohne diese Verzögerung wäre eine Message auf dem Weg zum Ziel-Host für einen Menschen zu schnell unterwegs. Die innerste If-Verzweigung stellt die eigentliche Prüfung dar. Hier wird getestet, ob der Empfangsport kein Blocking-Port ist und er sich im Forwarding-Status befindet. Dies sind die notwendigen Kriterien, die ein Port erfüllen muss, um eine Host-Nachricht empfangen zu dürfen. Geblockte Ports dürfen keine Messages empfangen, da sie einen redundanten Pfad blockieren und somit eine vorhandene Schleife aufbrechen. Das zweite Prüfungskriterium ist relevant, damit Nachrichten nur von Switches ausgewertet und weitergeleitet werden, wenn der Empfangsport nicht gerade eine BPDU auswertet. Wenn die If-Bedingungen zutreffen, wird die Message aus der PriorityBlockingQueue inbox3 ausgelesen und dort gleichzeitig entfernt. Nun beginnt die Auswertung dieser Nachricht, was in Listing 26 zu erkenne ist. Sollte der Empfangsport jedoch ein Blocking-Port sein, wird die Nachricht aus der Liste gelöscht und die Wire orange eingefärbt.

```
aktuellerSwitch.port_message_empfangen.wire.message_aktuell = false;
aktuellerSwitch.port_message_empfangen.wire.message_besucht = true;
// Wenn Host-Nachricht ueber einen Designated-Port empfangen wird, merke
// Sender-Adresse
if (aktuellerSwitch.port_message_empfangen.art == Portart.DESIGNATED_PORT) {
   aktuellerSwitch.sat_mac_table.put(message1.sender_mac_address,
         aktuellerSwitch.port_message_empfangen);
}
// Wenn Host-Nachricht ueber den Root-Port empfangen wird, darf
// Sender-Adresse nur gemerkt werden, wenn sie neu ist, da es ja sein
// kann, dass die Nachricht vom Root-Switch zurueckgeschickt wurde
else if (aktuellerSwitch.port_message_empfangen.art == Portart.ROOT_PORT) {
   if (!aktuellerSwitch.sat_mac_table
         .containsKey(message1.sender_mac_address)) {
      aktuellerSwitch.sat_mac_table.put(message1.sender_mac_address,
            aktuellerSwitch.port_message_empfangen);
   }
}
```

Listing 26: Auswerten einer Message

Die ersten beiden Zeilen dienen dem orangenen Einfärben der Wire, über die die Message eingetroffen ist. Wie im vorherigen Unterkapitel beschrieben, darf ein Switch nur in bestimmten Fällen einen Eintrag für den Sender-Host in seiner SAT-MAC-Table machen. Wenn der Empfangsport ein Designated-Port ist, fügt der Switch immer die Mac-Adresse des Sender-Hosts mit dem Empfangsport in die SAT-MAC-Table ein. Da diese Table als HashMap realisiert wurde, wird jedem vorhandenen Host der entsprechende Port zugewiesen. Falls der Empfangsport jedoch ein Root-Port ist, muss unbedingt geprüft werden, ob es bereits einen Eintrag zu dem Sender-Host in der HashMap gibt. Wenn dem so ist, war die Message bereits schon mal beim Switch und kommt gerade vom Root-Switch zurück. Dann darf kein neuer Eintag erstellt werden.

Die eben erwähnte Unterscheidung welcher der beiden Portarten der Empfangsport angehört, spielt auch bei der folgenden Auswertung ein entscheidende Rollte. In Listing 27 wird zuerst der Fall betrachtet, dass die Message von einem Designated-Port empfangen wurde.

```
message1.message_age = message1.message_age + 1;
if (message1.message_age < message1.max_age) {
   // Wenn Host-Nachricht ueber einen Designated-Port empfangen wird,
   // muss sie an Root-Switch weitergeleitet werden
   if (aktuellerSwitch.port_message_empfangen.art == Portart.DESIGNATED_PORT) {
      // Wenn aktueller Switch nicht Root-Switch ist
      if (aktuellerSwitch.root_switch == false) {
         // Suche Root-Port
         for (int i = 0; i < ports.length; i++) {
            // Wenn Root-Port gefunden, schicke Host-Nachricht weiter
            if (ports[i].art == Portart.ROOT_PORT) {
               Message message2 = copy_message();
               ports[i].wire.message_aktuell = true;
               ports[i].sendMessage(message2);
            }
         }
      }
      // Wenn aktueller Switch Root-Switch ist
      else {
         // Wenn Root-Switch den Ziel-Host noch nicht kennt
         if (!aktuellerSwitch.sat_mac_table.containsKey(message1.ziel_mac_address)) {
            // Schicke ueber alle Designated-Ports Host-Nachricht raus
            for (int i = 0; i < ports.length; i++) {
               // Sender-Port muss Designated-Port sein und sich im
               // Forwarding-Status befinden
               if (ports[i].art == Portart.DESIGNATED_PORT
                     && ports[i].status == Portstatus.FORWARDING) {
                  Message message2 = copy_message();
                  ports[i].wire.message_aktuell = true;
                  ports[i].sendMessage(message2);
               }
            }
         }
         // Wenn Root-Switch den Ziel-Host bereits kennt
         else if (aktuellerSwitch.sat_mac_table.containsKey(message1.ziel_mac_address)) {
            Port sender_port = aktuellerSwitch.sat_mac_table.get(message1.ziel_mac_address);
            // Sender-Port muss sich im Forwarding-Status befinden
            if (sender_port.status == Portstatus.FORWARDING) {
               Message message2 = copy_message();
               sender_port.wire.message_aktuell = true;
               sender_port.sendMessage(message2);
            }}}}
```

Listing 27: Weiterleiten einer Message 1

Wenn die Nachricht zu alt ist, wird sie nicht weitergeleitet. Ansonsten findet wieder eine Unterscheidung zwischen den beiden Portarten des Empfangsports also Designated- und Root-Port statt. In Listing 27 wird die zuerst genannte Portart betrachtet. Wenn es ein Designated-Port ist, muss unterschieden werden, ob er zu einem normalen oder zu dem Root-Switch gehört. Wenn der auswertende Switch ein normaler Switch ist, wird die Message über den Root-Port an den Root-Switch geschickt. Sollte jedoch der aktuelle Switch der Root-Switch sein, schaut er in seiner SAT-MAC-Table nach, ob er den Ziel-Host bereits kennt. Dies ist in der unteren Hälfte des

Listings 27 zu sehen. In dem Fall, dass der Ziel-Host noch nicht bekannt ist, wird die Message mittels Broadcast über alle Designated-Ports, die sich im Forwarding-Status befinden, verschickt. In der anderen Situation, die im unteren Else-If-Block abgebildet ist, wird die Nachricht nur über den Port, der zum Ziel-Host führt gesendet.

Nun muss noch der Fall betrachtet werden, dass die Nachricht über einen Root-Port empfangen wurde. Dies ist in Listing 28 dargestellt.

```
// Wenn Host-Nachricht ueber Root-Port empfangen wird, kommt sie vom
// Root-Switch. Dann muss sie ueber einen oder alle Designated-Ports
// weitergeleitet werden.
else if (aktuellerSwitch.port_message_empfangen.art == Portart.ROOT_PORT) {
   // Wenn aktueller Switch den Ziel-Host noch nicht kennt
   if (!aktuellerSwitch.sat_mac_table.containsKey(message1.ziel_mac_address)) {
      for (int i = 0; i < ports.length; i++) {
         // Schicke Message ueber alle Designated-Ports raus
         if (ports[i].art == Portart.DESIGNATED_PORT
               && ports[i].status == Portstatus.FORWARDING) {
            Message message2 = copy_message();
            ports[i].wire.message_aktuell = true;
            ports[i].sendMessage(message2);
         }
      }
   }
   // Wenn aktueller Switch den Ziel-Host bereits kennt
   else if (aktuellerSwitch.sat_mac_table.containsKey(message1.ziel_mac_address)) {
      Port sender_port = aktuellerSwitch.sat_mac_table.get(message1.ziel_mac_address);
      // Sender-Port muss sich im Forwarding-Status befinden
      if (sender_port.status == Portstatus.FORWARDING) {
         Message message2 = copy_message();
         sender_port.wire.message_aktuell = true;
         sender_port.sendMessage(message2);
      }}}
```

Listing 28: Weiterleiten einer Message 2

Wie in Listing 28 zu erkennen, muss diesmal nicht unterschieden werden, ob der auswertende Switch ein normaler oder der Root-Switch ist, da nur erstere über einen Root-Port verfügen. Die Überprüfung, ob der Ziel-Host bereits in der SAT-MAC-Table des aktuellen Switches vorhanden ist, kann jedoch nicht weggelassen werden. Wenn der Ziel-Host unbekannt ist, wird auch diesmal die Message mittels Broadcast über alle Designated-Ports verschickt, die sich im Forwarding-Status befinden. Falls das Ziel bekannt ist, wird die Nachricht nur über den entsprechenden Port weitergeleitet, der aber auch die eben erwähnten Kriterien erfüllen muss.

Das Weiterleiten einer Message beim Rapid Spanning Tree funktioniert sehr ähnlich und wird deshalb hier nicht näher betrachtet. Das Einzige was dabei noch berücksichtigt werden muss, sind die Edge-Ports, die genauso wie die Designated-Ports behandelt werden. Das bedeutet, dass wenn ein Switch eine Message über einen Edge-Port empfängt, diese auch an den Root-Switch weitergeleitet wird und wenn ein Switch eine Nachricht über den Root-Port bekommt, sie auch über die Edge-Ports verschickt wird.

9 Senden einer Nachricht in einem Netzwerk ohne Spannbaum

Um die Notwendigkeit der Spanning Tree Algorithmen zu verdeutlichen, können sie im Simulator abgeschaltet werden. Nach dem Starten der Simulation hat der Anwender nur die Möglichkeit Messages von einem Host zu einem anderen zu verschicken. Wie das funktioniert und wie sich die nicht blockierten Schleifen des vorliegenden Netzwerks darauf auswirken, wird in diesem Kapitel veranschaulicht. Zur besseren Übersicht ist dieses Kapitel in zwei Unterkapitel untergliedert, in denen die Funktionsweise und die Implementation getrennt voneinander erklärt werden.

9.1 Arbeitsweise des Nachrichtensendens ohne Spannbaum

Die Ausgangssituation ist ein gestartetes Netzwerk mit nicht blockierten Schleifen. Zur besseren Vergleichbarkeit wird dasselbe Beispiel wie in Kapitel 8 verwendet, das in Abbildung 65 dargestellt ist.

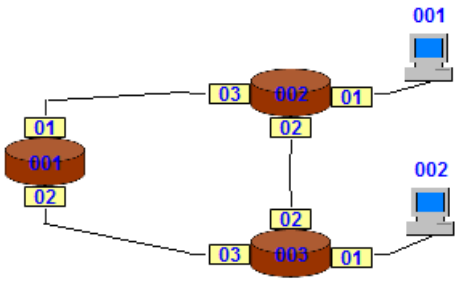


Abbildung 65: Ausgangssituation des Netzwerks zum Verschicken von Messages

Nachdem das vorliegende Netzwerk in den Simulator geladen oder manuell erstellt wurde, kann der Nutzer den RadioButton „None“ des in Kapitel 3 beschriebenen Steuerungsbereichs anwählen. Dann muss er sich entscheiden, ob er mittels der Checkbox die Alterung der SAT-MAC-Table-Einträge an- oder abschalten möchte. Standardmäßig ist sie aktiviert. Danach kann er die Simulation starten. Anders als im Kapitel 8 ist hier eine Alterung sinnvoll, da durch das ständige Zirkulieren der Messages im Netzwerk eventuell falsche Einträge in der SAT-MAC-Table entstehen können. Um zu verdeutlichen, dass die Simulation läuft und die Initialisierung aller Switches, Ports und Hosts abgeschlossen ist, werden ihre Beschriftungen blau eingefärbt. Diese Farbe wurde gewählt, weil sie in den Kapiteln 6 und 7 für die Designated-Komponenten also für die Standart-Komponenten eingesetzt wird.

Wenn der Anwender eine Message von Host 001 noch Host 002 verschicken möchte, verwendet er dazu den Kontextmenü-Eintrag „Send Message to“, bei dem er sich den gewünschten Ziel-Host aussuchen kann. Im vorliegenden Beispiel bleibt ihm nur die Möglichkeit die Nachricht an Host 002 zu schicken. Wie solch eine Message aufgebaut ist, wurde bereits im Kapitel 8 erklärt. Nachdem sie erzeugt wurde, übergibt Host 001 die Nachricht an die angeschlossene Wire, was in Abbildung 66 zu erkennen ist.

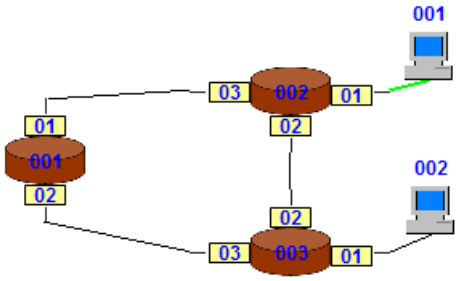


Abbildung 66: Message wurde von Host 001 an Wire übergeben

Da Switch 002 über keine verschiedenen Ports verfügt, erübrigt sich die Prüfung, ob die Nachricht von einem Root- oder Designated-Port und ob sie von einem normalen oder dem Root-Switch empfangen wurde. Wenn die Nachricht noch nicht zu alt ist, schaut der Switch nach, ob es bereits einen Eintrag vom Sender-Host in der SAT-MAC-Table gibt. Falls dem nicht so ist, erstellt er einen entsprechenden und startet seine Alterung. Im anderen Fall muss zusätzlich geprüft werden, ob auch der Empfangsport übereinstimmt. Bei erfolgreicher Prüfung wird die Alterung zurückgesetzt. Sollte jedoch die Message über einen anderen Port hereinkommen, darf die Alterung nicht zurückgesetzt werden. Kennt der Switch den Ziel-Host noch nicht, wird die Message mittels Broadcast verschickt. Dabei wird jedoch der Empfangsport übersprungen, da es sinnlos ist, die Nachricht zurück an den Sender zu schicken, und zwar wegen den folgenden zwei Gründen:

- Wenn der Sender weiß über welchen Port er den Ziel-Host erreichen kann, hat er ganz gezielt die Nachricht an den aktuellen Switch geschickt. Dann braucht der Empfänger sie auch nicht wieder zurückzuschicken, weil dies dann der falsche Weg ist.

- Wenn der Sender nicht weiß über welchen Port er den Ziel-Host erreichen kann, hat er einen Broadcast über all seine Ports gemacht. Also braucht der aktuelle Switch die Nachricht auch nicht zurückzuschicken, da über alle Ports des Sender-Switches bereits die Message verschickt wurde.

Das Einfärben der Leitungen funktioniert identisch wie in Kapitel 8 beschrieben. Die Leitungen, auf denen sich gerade eine Message befindet, werden grün dargestellt und die Wires, die von einer Nachricht wieder verlassen wurden, werden orange eingefärbt.

In den folgenden Abbildungen wird gezeigt, wie solch eine Nachricht weitergeleitet wird. Nachdem Switch 002 die Message ausgewertet hat, wird sie über die Ports 02 und 03 versendet. Dies ist in Abbildung 67 dargestellt.

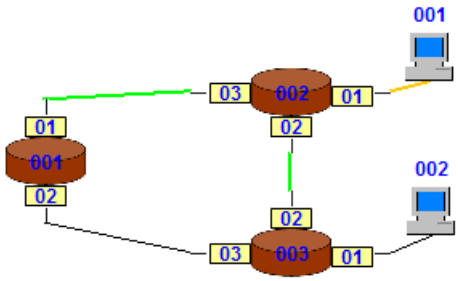


Abbildung 67: Switch 002 verschickt Message mittels Broadcast

Die Switches 001 und 003 empfangen jetzt ihrerseits die Message, werten sie aus und leiten sie mittels Broadcast weiter. Das bedeutet, dass Switch 001 die Nachricht über Port 02 und Switch 003 die Nachricht über Port 01 und 03 versendet. Dies ist in Abbildung 68 zu sehen.

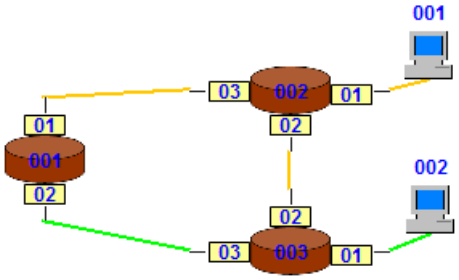


Abbildung 68: Switch 001 und 003 leiten Message mittels Broadcast weiter

Nun verarbeitet Switch 001 die Message, die er über Port 02 empfängt und schickt sie ebenfalls mittels Broadcast über Port 01 raus. Switch 003 verfährt genauso und leitet die Nachricht über Port 01 und 02 weiter. In Abbildung 69 ist dieser Schritt zu erkennen.

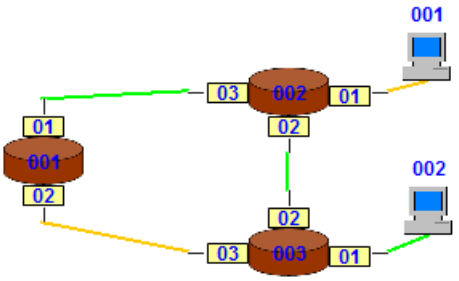


Abbildung 69: Switch 001 und 003 leiten Message mittels Broadcast weiter

Wie abzusehen, zirkuliert die Message immer weiter im Netzwerk, bis sie ihr maximales Alter erreicht hat und gelöscht wird. Der finale Zustand ist in Abbildung 70 erkennbar.

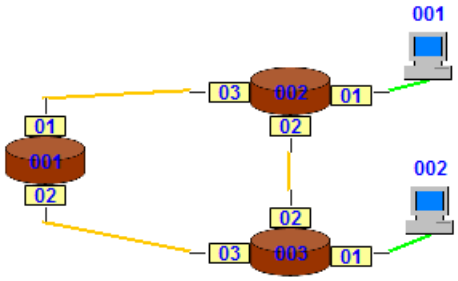


Abbildung 70: Finaler Zustand

Zur Veranschaulichung des gelernten Wissens folgt nun noch Abbildung 71.

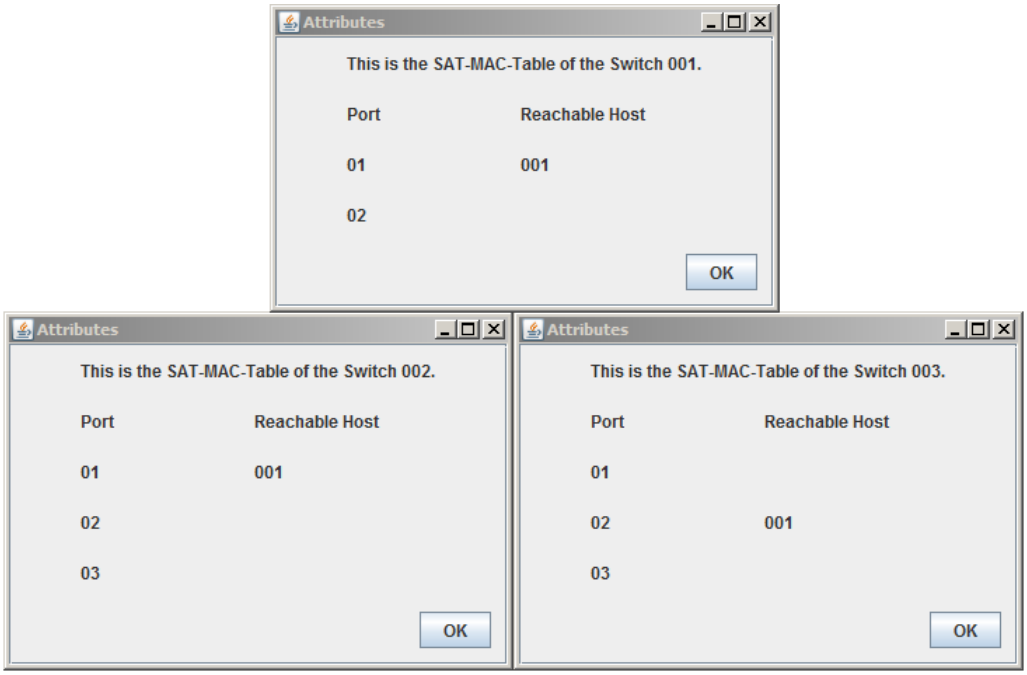


Abbildung 71: SAT-MAC-Tables nach Message von Host 001 an Host 002

Hier ist zu sehen, über welchen Port der jeweilige Switch Host 001 erreichen kann.

Wenn der Nutzer nun von Host 002 eine Message an Host 001 schickt, wissen alle Switches wo dieser zu erreichen ist und leiten deshalb die Nachricht gezielt weiter. In Abbildung 72 ist der finale Zustand nach dem Verschicken der zweiten Message zu erkennen.

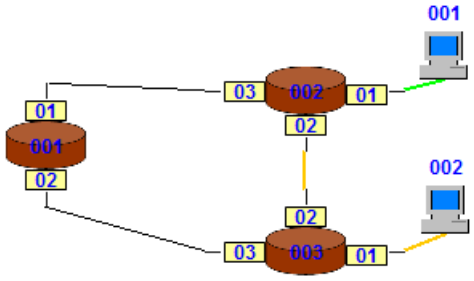


Abbildung 72: Finaler Zustand

Da die Message von Host 002 nach Host 001 nicht über Switch 001 gelaufen ist, weiß dieser nicht, wo Host 002 zu erreichen ist. Zur Veranschaulichung folgt Abbildung 73.

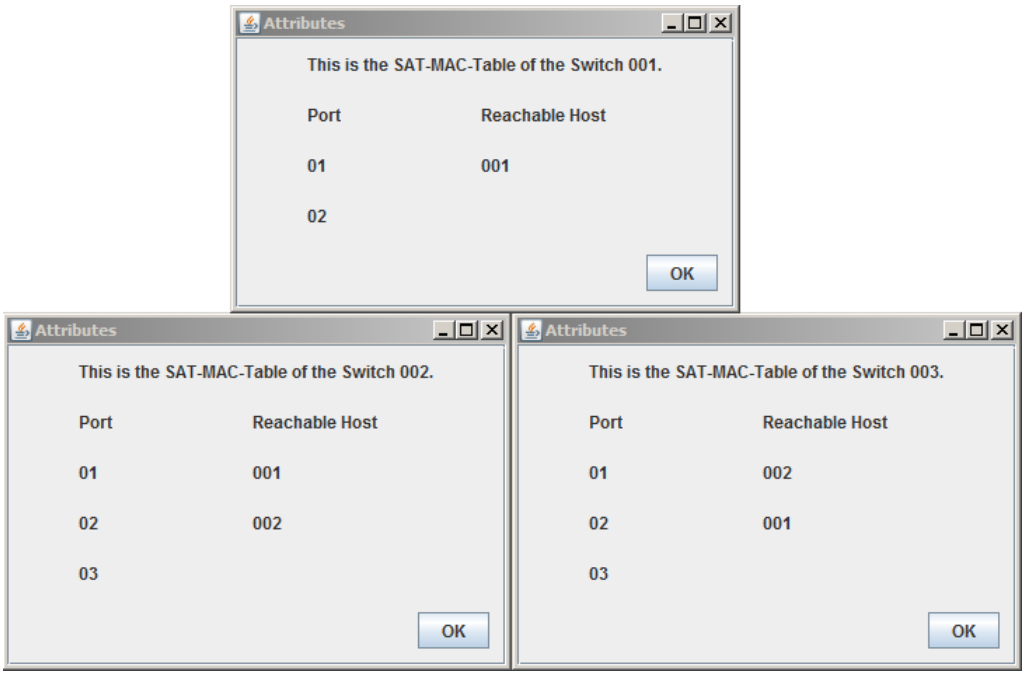


Abbildung 73: SAT-MAC-Tables nach Message von Host 002 an Host 001

Zu Beginn dieses Unterkapitels wurde erwähnt, dass die Einträge in den SAT-MAC-Tables bei einem Netzwerk ohne Spanning Tree Algorithmus altern. Das maximal erlaubte Alter wurde willkürlich auf 2 Minuten gesetzt.

9.2 Implementation des Nachrichtensendens ohne Spannbaum

Nachdem im vorherigen Unterkapitel ausführlich erklärt wurde, wie der Nutzer Messages von einem Host zu einem anderen verschicken kann, folgen nun einige Erläuterungen zur Implementation dieser Funktionalität. Auf den Aufbau und das Erzeugen der Messages wird hier nicht nochmals eingegangen, da dies bereits in Kapitel 8 abgehandelt wurde. Da jeder Switch als Thread realisiert wurde, der vom Start bis zum Beenden der Simulation ununterbrochen aktiv ist, überprüft er kontinuierlich, ob einer der Einträge in der SAT-MAC-Table zu alt geworden und ob eine Host-Nachricht eingetroffen ist. Die erste Prüfung erfolgt mit der folgenden Methode, die in Listing 29 zu sehen ist.

```
public void loesche_alte_eintraege(Switch aktuellerSwitch, Port[] ports) {
   for (String key : aktuellerSwitch.alterung_eintraege.keySet()) {
      if (aktuellerSwitch.alterung_eintraege.get(key) + 120 * 1000 < System
            .currentTimeMillis()) {
         aktuellerSwitch.alterung_eintraege.remove(key);
         aktuellerSwitch.sat_mac_table.remove(key);
      }
   }
}
```

Listing 29: Alterung der Einträge der SAT-MAC-Table

Da die SAT-MAC-Table als HashMap realisiert wurde, deren Einträge aus der Mac-Adresse des Sender-Hosts und dem Empfangsport bestehen, kann hier der Zeitpunkt, zu dem der Eintrag erstellt wurde, nicht auch noch gespeichert werden. Aus diesem Grund wurde eine zweite HashMap verwendet, die als Key auch diesmal die Mac-Adresse des Sender-Hosts und als Value die Systemzeit beim Erstellen des Eintrags benutzt. Nun kann einfach über die zweite HashMap iteriert und mittels einfacher If-Verzweigung geprüft werden, ob das maximal erlaubte Alter von 2 Minuten überschritten ist. Wenn dem so ist, werden die entsprechenden Einträge aus beiden HashMaps entfernt. Alle empfangenen Messages werden genauso wie die BPDUs aus den Kapiteln 6 und 7 in einer PriorityBlockingQueue gespeichert, bis sie zur Auswertung entnommen werden. Auch hier wird jede Host-Nachricht drei Sekunden in dieser Liste liegen gelassen, bevor sie ausgewertet und weitergeleitet wird. Dies dient wie in Kapitel 8 bereits erwähnt dazu, dass der Nutzer die Nachrichten bei ihrem Weg durch das Netzwerk besser verfolgen kann.

Sobald bei einer der wartenden Messages die Zeit verstrichen ist, wird sie ausgewertet. Der erste Schritt ist die Prüfung des Alters der Nachricht. Dieses entspricht der Anzahl der passierten Switches. Wenn sie zu alt ist, wird sie gelöscht und nicht weitergeleitet.

Nun beginnt das Lernen des Sender-Hosts, was in Listing 30 dargestellt ist.

```
// Wenn Sender-Host noch nicht in SAT-MAC-Table enthalten ist, fuege
// Eintrag hinzu und starte Alterung
if (!aktuellerSwitch.sat_mac_table.containsKey(message1.sender_mac_address)) {
   aktuellerSwitch.sat_mac_table.put(message1.sender_mac_address,
         aktuellerSwitch.port_message_empfangen);
   aktuellerSwitch.alterung_eintraege.put(message1.sender_mac_address,
         System.currentTimeMillis());
}
// Wenn Sender-Host bereits in SAT-MAC-Table enthalten ist
else {
   // Wenn Empfangsport mit gespeichertem Port uebereinstimmt, setze Alterung zurueck
   if (aktuellerSwitch.sat_mac_table.get(message1.sender_mac_address).id
         == ((Port) message1.empfaenger).id) {
      aktuellerSwitch.alterung_eintraege.put(message1.sender_mac_address,
            System.currentTimeMillis());
   }
}
```

Listing 30: Lernen des Sender-Hosts

Anders als im Kapitel 8 wird der Sender-Host nur dann gelernt, wenn er noch nicht in der SAT-MAC-Table vorkommt. Dies ist diesmal wichtig, da die vorhanden Schleifen im Netzwerk nicht blockiert werden und so dieselbe Message über mehrere Ports den Switch erreicht und die Ports nicht durch eine Portart unterschieden werden können. Deshalb darf der Sender-Host nur beim erstmaligen Empfang gespeichert werden.

Der Fall, dass es noch keinen Eintrag zum Sender-Host gibt, ist im Then-Teil dargestellt. Dort werden für ihn Einträge in beiden HashMaps erstellt. Wenn es bereits Einträge gibt, muss geprüft werden, ob auch der Empfangsport übereinstimmt. Falls dem so ist, wird in der zweiten HashMap die gespeicherte Systemzeit durch die aktuelle ersetzt, was einem Zurücksetzten des Alterungs-Timers entspricht.

Nach dem Lernen des Sender-Hosts folgt das Weiterleiten der Message, das in Listing 31 dargestellt ist. Auch hier erfolgt nur eine Prüfung, ob es in der SAT-MAC-Table bereits einen Eintrag vom Ziel-Host gibt. Der Then-Teil repräsentiert den negativen und der Else-Teil den positiven Fall. Sollte der Ziel-Host noch unbekannt sein, verschickt der Switch mittels Broadcast die Message über alle seine Ports, überspringt dabei aber den Empfangsport. Warum er diesen auslässt, wurde bereits im vorherigen Unterkapitel erklärt. Wenn der Ziel-Host schon bekannt ist, wird nur über den einen Port, der zum Host führt, die Nachricht weitergeleitet.

```
// Wenn Switch den Ziel-Host noch nicht kennt
if (!aktuellerSwitch.sat_mac_table.containsKey(message1.ziel_mac_address)) {
   // Schicke ueber alle Ports Host-Nachricht raus
   for (int i = 0; i < ports.length; i++) {
      // Ueberspringe Empfangsport
      if (ports[i].id != aktuellerSwitch.port_message_empfangen.id) {
         Message message2 = copy_message();
         ports[i].wire.message_aktuell = true;
         ports[i].sendMessage(message2);
      }
   }
}
// Wenn Switch den Ziel-Host bereits kennt
else if (aktuellerSwitch.sat_mac_table.containsKey(message1.ziel_mac_address)) {
   Port sender_port = aktuellerSwitch.sat_mac_table.get(message1.ziel_mac_address);
   Message message2 = copy_message();
   sender_port.wire.message_aktuell = true;
   sender_port.sendMessage(message2);
}
```

Listing 31: Weiterleiten einer Message

10 Erstellen von Netzwerken im Spanning Tree Simulator

Als letzte große Erweiterung wird dem Nutzer die Möglichkeit gegeben, innerhalb des Simulators Netzwerke selbst zu erstellen oder bereits geladene Netze zu verändern. Dies alles kann direkt im Bereich zur graphischen Anzeige der Netzwerke erledigt werden. Zur besseren Übersicht ist dieses Kapitel in zwei Unterkapitel untergliedert, in denen die Funktionsweise und die Implementation getrennt voneinander erklärt werden.

10.1 Arbeitsweise der Netzwerkerstellung

In der Bachelorarbeit [Jan10b] konnte der Nutzer nach dem Start des Spanning Tree Simulators nur ein vorher in einem Configuration-File definiertes Netzwerk ins Programm laden. Diese Files sind in XML geschrieben, was das Erstellen eines Netzwerks ziemlich umständlich macht. Deshalb ist im Rahmen dieser Masterarbeit das Bauen von Netzwerken direkt im Simulator realisiert worden. Um die Bedienung möglichst einfach zu halten, können alle dazu notwendigen Aktionen über entsprechende Kontextmenüs aufgerufen werden.

Wenn der Anwender ein Netzwerk erstellen möchte, fängt er am besten mit den Switches an, da sie die zentralen Komponenten darstellen. Dazu ruft er das Kontextmenü des Panels zur Darstellung von Netzen auf. Nun hat er die Möglichkeit mittels der Schaltflächen „Create Switch“ und „Create Host“ beliebig viele Switches und Hosts zu platzieren. In diesem Beispiel werden drei Switches erzeugt, die in Abbildung 74 dargestellt sind.

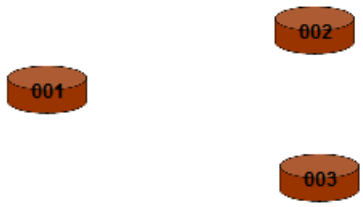


Abbildung 74: Erzeugung von drei Switches

Wie in Abbildung 74 zu erkennen, verfügen so erzeugte Switches noch über keine Ports, die erst mittels der Kontextmenüs der Switch-Komponenten erstellt werden müssen. Eine grundlegende Schwierigkeit beim Bauen von Netzwerken direkt im Simulator stellt das notwendige Wissen der einzelnen Bausteine dar. Beim Laden eines Configuration-Files hat der Nutzer dieses bereits in dem File selbst angegeben und so steht es dem Simulator beim Anlegen der Objekte zur Verfügung. Dies ist hier aber nicht der Fall, was bedeutet, dass die Wissens-Attribute wie z.B. das maximal erlaubte Alter einer Nachricht Default-Werte benötigen. Welche Attribute es gibt und welche Aufgabe sie haben, wurde bereits in den Kapiteln 2.2, 2.3, 6 und 7 ausführlich beschrieben. Bei den meisten dieser Werte gibt es keine Probleme, da sie nur aus einem konstanten Wert bestehen. Jedoch bei der Mac-Adresse und der ID sieht es anders aus, da sie für jede erzeugte Komponente separat generiert werden müssen. Prinzipiell beginnen die Werte für beide bei eins und werden bei jeder neuen Komponente um eins erhöht. Wie dies im Detail realisiert wurde, ist im folgenden Unterkapitel beschrieben.

Nach der Platzierung der drei Switches werden zwei Hosts erstellt, bei denen dieselbe Werte-Problematik vorliegt. Dies ist in Abbildung 75 zu sehen.

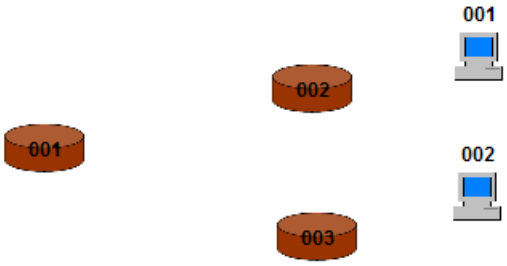


Abbildung 75: Erzeugung von zwei Hosts

Anschließend folgen die Ports der drei Switches. Sie werden über den Kontextmenü-Eintrag „Create Port“ des jeweiligen Switches generiert. Das Besondere hieran ist, dass der Nutzer sich zwischen vier möglichen Positionen entscheiden kann, und zwar sind dies im Uhrzeigersinn:

- top
- right
- bottom
- left

Diese Umsetzung bietet die Möglichkeit beim Aufbau eines Netzwerks eine der vier vorhandenen Position für den Port zu wählen. Jedoch hat dies den Nachteil, dass jeder Switch nur maximal vier Ports haben kann. Diese Einschränkung ist aber nicht so gravierend, da der Simulator hauptsächlich für den Lehrbetrieb entwickelt wurde und somit nur relativ kleine Netzwerke verwendet werden. Wenn der Anwender unbedingt mehr als vier Ports haben möchte, muss er diese in dem Configuration-File anlegen. Dort können beliebig viele Ports pro Switch definiert werden, die beim Laden auch richtig behandelt und angelegt werden. Jedoch überlagern sie sich dann bei der Darstellung des Netzwerks. Wie in verschiedenen vorherigen Kapiteln beschrieben, können mit den Default-Einstellungen der beiden Spanning Tree Algorithmen nur Netzwerke betrieben werden, deren Switches mit maximal vier anderen Switches verbunden sind. Andernfalls wäre das Nachrichtenaufkommen zu groß, um noch verarbeitet werden zu können. Um solche großen Netze zu verwenden, müssen verschiedene Werte wie beispielsweise das maximale Alter eine Nachricht, die Hello-Time und das Timeout-Intervall entsprechend angepasst werden. Unerfahrene Anwender sollten sich mit den vier Ports pro Switch begnügen, da sie auch so eine Vielzahl von verschiedenen Netzwerken kreieren können, an denen die Funktionsweise und die Vorteile der Spanning Tree Algorithmen zu sehen sind.

Die Abbildung 76 zeigt das bisherige Netzwerk mit allen generierten Ports.

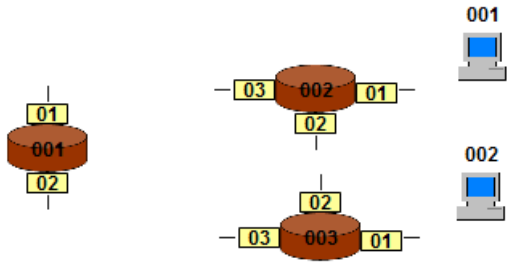


Abbildung 76: Erzeugung von Ports

Im letzten Schritt müssen nun noch alle Hosts und Ports miteinander verbunden werden. Dazu werden die Kontextmenü-Einträge „Connect to Host“ und „Connect to Switch“ der Ports benötigt. Direkt nach dem Anklicken eines dieser Menüpunkte wird das entsprechende Wire-Objekt angelegt und die Linie gezeichnet. Es wurde hierbei darauf geachtet, dass ein Port eines Switches nicht mit einem anderen Port desselben Switches verbunden werden kann. Eine weitere grundlegende Richtlinie ist, dass jeder Port und jeder Host nur mit genau einer Wire verbunden werden dürfen, was hier auch berücksichtigt wurde. In Abbildung 77 ist das fertige Netzwerk zu erkennen.

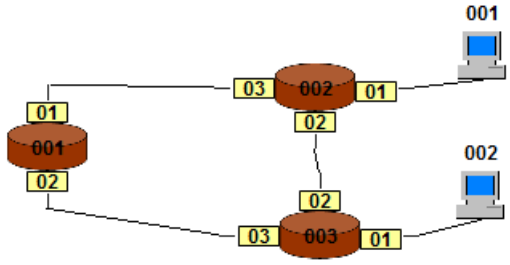


Abbildung 77: Fertige Netzwerk

Die Simulation kann jedoch erst gestartet werden, wenn alle folgenden Kriterien erfüllt sind:

- Es muss mindestens zwei Switches geben.
- Jeder Switch muss über mindestens einen Port verfügen.
- Jeder Port muss mit einer Wire verbunden sein.
- Jeder Host muss an genau eine Wire angeschlossen sein.
- Jede Wire muss einen Anfang und ein Ende haben.

Ein Netzwerk muss also aus mindestens zwei Switches mit je einem Port, die durch eine Wire miteinander verbunden sind, bestehen. Hosts hingegen werden nicht benötigt, um die Simulation

starten zu können. Die Möglichkeit das Netz in einem Configuration-File zu speichern oder zu erweitern, besteht zu jeder Zeit, außer während einer laufenden Simulation.

Nun wurde ausführlich beschrieben, wie ein Netzwerk direkt im Simulator zusammengebaut werden kann. Die Möglichkeit Netzwerke, die mittels einem Configuration-File ins Programm geladen wurden, nachträglich zu erweitern, wurde auch realisiert. Um es zu starten, muss es dieselben Kriterien erfüllen.

Beim Bauen oder Erweitern eines Netzwerks ist nicht auszuschließen, dass der Nutzer hin und wieder einen Fehler begeht, indem er eine Komponente erzeugt, die er eigentlich gar nicht haben möchte. Deshalb ist es notwendig die einzelnen Bausteine wieder entfernen zu können. Dies geschieht mittels der Kontextmenü-Einträge „Delete Switch“, „Delete Port“ und „Delete Host“ der entsprechenden Komponenten. Wenn ein nicht verbundener Port oder Host gelöscht wird, werden die dargestellte Komponente und das dazugehörige Objekt aus allen Listen entfernt. Schwieriger wird es, wenn eine verbundene Komponente oder ein ganzer Switch gelöscht werden soll. Im ersten Fall muss zusätzlich zur Darstellung und dem Objekt auch noch die verbundene Wire entfernt werden. Da eine Wire immer eine Anfangs- und ein End-Komponente hat, die sich das Wire-Objekt merken, muss auch das gegenüberliegende Objekt angepasst werden. Wenn der Anwender einen ganzen Switch löschen will, müssen zusätzlich zum eigentlichen Switch noch alle dazugehörigen Ports und die angeschlossenen Wires entfernt werden. Auch hier müssen alle gegenüberliegenden Komponenten angepasst werden. Um ein versehentliches Löschen zu verhindern, wurde eine Sicherheitsabfrage eingebaut.

Außer dem Löschen einzelner Bestandteile eines Netzwerks, gibt es noch die Möglichkeit, ohne den Simulator neu starten zu müssen, ein neues Netzwerk zu erstellen. Dies geschieht über den Kontextmenü-Eintrag „New Network“ des Panels zur Darstellung eines Netzes. Auch hier erfolgt die eben erwähnte Sicherheitsabfrage.

10.2 Implementation der Netzwerkerstellung

Nachdem im vorherigen Unterkapitel ausführlich erklärt wurde, was es für Möglichkeiten rund um das Erstellen eines Netzwerks direkt im Simulator gibt und wozu sie dienen, wird hier beschrieben, wie dies implementiert wurde.

Auch diesmal wird zuerst das Erstellen eines Switches erläutert. Der erste Schritt dabei ist das Erzeugen eines Switch-Objekts, das mit allen notwendigen Informationen gefüllt wird. Exemplarisch wird in Listing 32 das Bestimmen der Mac-Adresse dargestellt, da dies zusammen mit der ID zum schwierigsten Aspekt gehört.

```
// Bestimme Mac-Adresse
boolean test_mac = false;
int mac1 = 1;
// Wenn es bereits mindestens einen Switch in Liste gibt
if (switch_map.size() > 0) {
   // Arbeite solange, bis eine noch nicht vergebene Mac-Adresse
   // gefunden wurde
   while (test_mac == false) {
      for (String key : switch_map.keySet()) {
         int mac_vorhanden = new Integer(key);
         // Wenn aktuelle Mac-Adresse bereits vergeben wurde,
         // verlasse innere Schleife und pruefe naechste moegliche Zahl
         if (mac1 == mac_vorhanden) {
            test_mac = false;
            break;
         }
         // Wenn eine neue Mac-Adresse gefunden wurde
         else {
            test_mac = true;
         }
      }
      // Wenn eine neue Mac-Adresse gefunden wurde, verlasse aeussere Schleife
      if (test_mac == true) {
         break;
      }
      // Wenn aktuelle Mac-Adresse bereits vergeben wurde, erhoehe Zahl um 1
      mac1 = mac1 + 1;
   }
}
```

Listing 32: Bestimmen der Mac-Adresse eines Switches

Der zentrale Aspekt sind die beiden ineinander verschachtelten Schleifen, die die erste noch nicht vergebene Mac-Adresse suchen. Diese Adressen beginnen normalerweise bei 1 und werden bei jedem weiteren Switch um 1 erhöht. Da aber wie vorher beschrieben auch einzelne Komponenten gelöscht werden können, kommt es zwangsläufig zu Lücken bei den vergebenen Adressen. Wenn ein neuer Switch erzeugt wird, soll immer die kleinst mögliche Adresse genommen werden, damit diese Lücken wieder aufgefüllt werden. Genau dies wurde mit dem angegebenen Quellcode realisiert. Die äußere While-Schleife ist solange aktiv, bis eine noch nicht vergebene Mac-Adresse gefunden wurde und die innere For-Schleife durchläuft das Port-Array. In der Variablen mac1 steht die angenommene Mac-Adresse, die bei 1 beginnt. Bei jedem Durchlauf der For-Schleife wird geschaut, ob es bereits einen Port mit der angenommene Adresse gibt. Wenn dem so ist,

wird die innere Schleife abgebrochen und die angenommen Adresse um eins erhöht. Sobald eine noch nicht vergebene Mac-Adresse gefunden wurde, werden beide Schleifen sofort verlassen. Danach wird sie dem Switch-Objekt zugewiesen. Die Bestimmung der ID funktioniert genauso, weshalb auf sie hier nicht weiter eingegangen wird. Am Ende wird das Objekt in einer Hash-Map gespeichert. Bei der Visualisierung der Switch-Komponente hat sich bis auf ein paar neue Kontextmenü-Einträge nichts geändert.
Auf das Erzeugen der Host-, Port- und Wire-Objekte und die Visualisierung der Host- und Wire-Komponente wird ebenfalls nicht näher eingegangen, da dies sehr ähnlich funktioniert bzw. sich daran im Vergleich zu Bachelorarbeit [Jan10b] nichts geändert hat.

Als einziges hat sich die Implementation der Darstellung einer Port-Komponente verändert. Dies ist in Listing 33 zu erkennen. Es gibt nur vier mögliche Positionen, die ein Port einnehmen kann, und zwar oben, rechts, unten und links. In der Bachelorarbeit [Jan10b] wurde die Position eines Ports durch seine ID bestimmt. Also wenn ein Port die ID 1, 5, 9, usw. hatte, wurde er immer oberhalb des Switches plaziert und bei der ID 2, 6, 10, usw. rechts. Bei den beiden verbleibenden Möglichkeiten funktionierte es analog. Dies wurde im Rahmen der Masterarbeit geändert, weil der Nutzer selbst die Option bekommen sollte, die Position zu wählen. Um dies zu realisieren gibt es die vier im vorherigen Unterkapitel erwähnten Kontextmenü-Punkte einer Switch-Komponente. Wie im Listing 33 zu sehen, wird die Position nun über den Namen des angeklickten Kontextmenü-Eintrags bestimmt.

```
// Port links
if (aport.port_position.equals("left")) {
   label_Port.setLocation(aswitch.position_x - 28,
         aswitch.position_y + 8);
   ...
   aport.position_x = aswitch.position_x - 38;
   aport.position_y = aswitch.position_y + 14;
   ...
}
// Port oben
else if (...){
...
}
...
```

Listing 33: Visualisierung der Port-Komponenten

Nach der Erzeugung jeder Komponente findet eine Überprüfung statt, ob das Netzwerk bereits fertig ist. Erst bei erfolgreicher Prüfung, wird der Start-Button aktiviert. Dies ist in Listing 34 zu sehen.

```
if (switch_map != null) {
   for (String key : switch_map.keySet()) {
      Switch test_switch = switch_map.get(key);
      // Es muss mindestens ein Port pro Switch vorhanden sein
      if (test_switch.ports.length > 0) {
         for (int i = 0; i < test_switch.ports.length; i++) {
            // Wenn Port mit Wire verbunden ist
            if (test_switch.ports[i].wire != null) {
               // Wenn Port ein Wire-Objekt hat, muss geprueft
               // werden, ob dieses auch einen Anfang und ein Ende hat
               if (test_switch.ports[i].wire.start != null
                     && test_switch.ports[i].wire.end != null) {
                  fertig_switch = true;
               }
               // Wenn Port ein Wire-Objekt hat, das entweder
               // keinen Start- oder End-Punkt hat
               else {
                  fertig_switch = false;
                  break;
               }
            }
            // Wenn es einen Port gibt, der noch mit keiner Wire
            // verbunden ist
            else {
               fertig_switch = false;
               break;
            }
         }
      }
      // Wenn es einen Switch gibt, der noch keinen Port hat
      else {
         fertig_switch = false;
         break;
      }
      // Wenn Netzwerk noch nicht fertig aufgebaut ist
      if (fertig_switch == false) {
         break;
      }
   }
} else {
   fertig_switch = false;
}
```

Listing 34: Test auf Vollständigkeit eines Netzwerks

Aus Platzgründen wird hier nur Überprüfung der Switches und Ports gezeigt. Die beiden ineinander verschachtelten For-Schleifen iterieren über alle Switches und alle Ports. Dabei werden nacheinander die im vorherigen Unterkapitel genannten Kriterien betrachtet und getestet, ob sie erfüllt sind oder nicht. Wenn ein Test fehlschlägt, gilt das Netzwerk als unfertig und die Simulation kann nicht gestartet werden.

Bei der Betätigung des Kontextmenü-Punkts „New Network“ des Panels zur Darstellung des Netzwerks wird er Simulator neu initialisiert. Das Löschen einzelner Bestandteile geschieht über das Kontextmenü der zu löschenden Komponente. Da dies beim Switch am schwierigsten ist, wird dieser Fall hier betrachtet. Im Listing 35 ist der dazugehörige Quellcode abgebildet.

```
JLabel switch_label = aswitch.label_switch;
// Wenn Switch ueber Ports verfuegt
if (aswitch.ports.length > 0) {
   for (int j = 0; j < aswitch.ports.length; j++) {
      JLabel port_label = aswitch.ports[j].label_port;
      JLabel line_label = aswitch.ports[j].label_line;
      // Wenn Port an Wire angeschlossen ist
      if (aswitch.ports[j].wire != null) {
         Wire switch_wire = aswitch.ports[j].wire;
         // Loesche Wire-Objekt aus gegenueberliegender Komponente
         // Wenn angeklickter Port Start-Port der Wire ist
         if (aswitch.ports[j].id == ((Port) switch_wire.start).id) {
            // Wenn Ende ein Host ist
            if (Host.class.isInstance(switch_wire.end)) {
               ((Host) switch_wire.end).wire = null;
            }
            // Wenn Ende ein Port ist
            else {
               ((Port) switch_wire.end).wire = null;
            }
         }
         // Wenn angeklickter Port End-Port der Wire ist
         else {
            ((Port) switch_wire.start).wire = null;
         }
         // Loesche Wire, die mit Host verbunden ist
         wire_map.remove(aswitch.ports[j]);
      }
      // Loesche Port-Label und Line-Label
      panel_netzwerk.remove(port_label);
      panel_netzwerk.remove(line_label);
   }
}
// Loesche Switch aus HashMap
switch_map.remove(aswitch.mac_address);
// Loesche Switch-Label
panel_netzwerk.remove(switch_label);
panel_netzwerk.updateUI();
test_netzwerk_fertig();
```

Listing 35: Löschen eines Switches

Zuerst wird geschaut, ob der zu löschende Switch überhaupt über Ports verfügt. Sollte dem so sein, muss geprüft werden, ob einige Ports bereits mit einer Wire verbunden sind. Wenn dies auch zutrifft, muss das Wire-Objekt bei der gegenüberliegenden Komponente gelöscht werden. Erst dann kann die Wire selbst entfernt werden. Anschließend wird der Port und zum Schluss der Switch selbst gelöscht.

11 Weitere Erweiterungen und Überarbeitungen des Simulators

In diesem Kapitel werden ein paar kleinere Erweiterungen des Spanning Tree Simulators vorgestellt.

11.1 Anzeige von empfangenen Configuration-Messages

Hier wird das überarbeitete Fenster zur Anzeige von empfangenen Configuration-Messages vorgestellt. Es wird wie schon in der Bachelorarbeit [Jan10b] durch die Buttons „Show all received BPDUs“ aufgerufen. Sie sind nur während einer laufenden oder nach einer beendeten Simulation sichtbaren. Durch Anklicken eines dieser Buttons öffnet sich ein Fenster, in dem alle empfangenen BPDUs eines Switches nach der Empfangszeit aufsteigend sortiert angezeigt werden. Daran hat sich auch im Rahmen dieser Masterarbeit nichts geändert. Jedoch die Art der Darstellung wurde grundlegend verändert. In der Bachelorarbeit wurde für jeden einzelnen Wert der Tabelle ein eigenes Label erzeugt, was sehr aufwendig und umständlich war. Jetzt wird die gesamte Darstellung mit einer JTable realisiert.

Außer an der Anzeige wurde auch einiges an der Funktionsweise verbessert. Jetzt ist es beispielsweise möglich die Größe des aufgerufenen Fensters nach Belieben zu verändern. Um die empfangenen Nachrichten der einzelnen Switches besser miteinander vergleichen zu können, kann der Nutzer die Fenster von mehreren Switches gleichzeitig aufrufen. Dies ist in Abbildung 78 zu sehen.

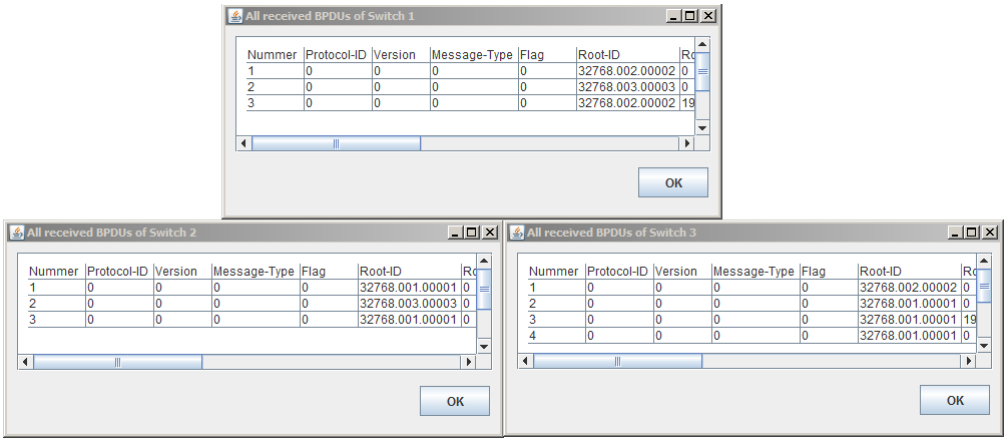
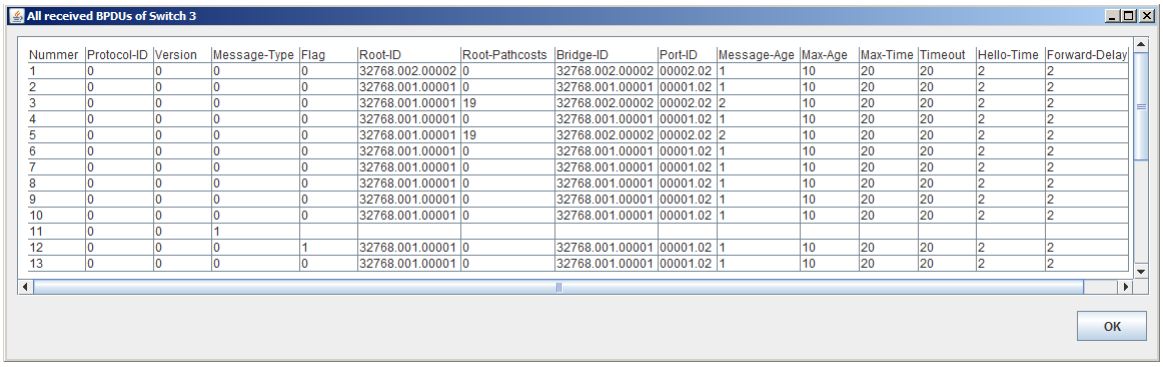


Abbildung 78: Die Fenster zur Anzeige von empfangenen Configuration-Messages

Ein weiterer Aspekt, der in dieser Masterarbeit berücksichtigt werden musste, waren die verschiedenen BPDU-Typen, die durch die Rekonfiguration des Spannbaums und durch den Rapid Spanning Tree Algorithmus hinzukamen. In Abbildung 79 sind die drei Typen des Spanning Tree Algorithmus IEEE 802.1D dargestellt.

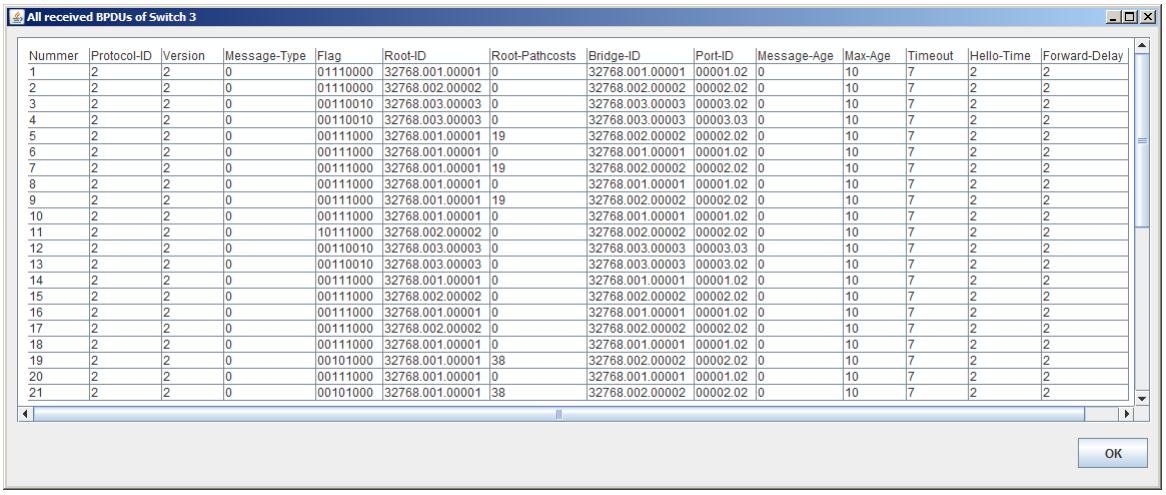


Nummer	Protocol-ID	Version	Message-Type	Flag	Root-ID	Root-Pathcosts	Bridge-ID	Port-ID	Message-Age	Max-Age	Max-Time	Timeout	Hello-Time	Forward-Delay
1	0	0	0	0	32768.002.00002	0	32768.002.00002	00002.02	1	10	20	20	2	2
2	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
3	0	0	0	0	32768.001.00001	19	32768.002.00002	00002.02	2	10	20	20	2	2
4	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
5	0	0	0	0	32768.001.00001	19	32768.002.00002	00002.02	2	10	20	20	2	2
6	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
7	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
8	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
9	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
10	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
11	0	0	1											
12	0	0	0	1	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2
13	0	0	0	0	32768.001.00001	0	32768.001.00001	00001.02	1	10	20	20	2	2

Abbildung 79: Fertiges Netzwerk

In den Zeilen 1-10 und 13 sind die ganz normalen BPDUs zu sehen, die vom Root-Switch entsprechend der Hello-Time erzeugt werden. Sie dienen, wie im Kapitel 2.2 beschrieben, zum Aufbauen des Spannbaums. In den Zeilen 11 und 12 ist eine TCN-BPDU und eine Antwort-BPDU zu sehen, die beim Ausfall einer Verbindungsleitung zur Rekonfiguration des Spannbaums benötigt werden. Diese Thematik wurde in Kapitel 6 ausführlich erklärt.

Beim Rapid Spanning Tree Algorithmus hingegen gibt es andere BPDU-Typen, was in den Kapiteln 2.3 und 7 erläutert wurde. Diese sind in Abbildung 80 dargestellt.



Nummer	Protocol-ID	Version	Message-Type	Flag	Root-ID	Root-Pathcosts	Bridge-ID	Port-ID	Message-Age	Max-Age	Timeout	Hello-Time	Forward-Delay
1	2	2	0	01110000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
2	2	2	0	01110000	32768.002.00002	0	32768.002.00002	00002.02	0	10	7	2	2
3	2	2	0	00110010	32768.003.00003	0	32768.003.00003	00003.02	0	10	7	2	2
4	2	2	0	00110010	32768.003.00003	0	32768.003.00003	00003.03	0	10	7	2	2
5	2	2	0	00111000	32768.001.00001	19	32768.002.00002	00002.02	0	10	7	2	2
6	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
7	2	2	0	00111000	32768.001.00001	19	32768.002.00002	00002.02	0	10	7	2	2
8	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
9	2	2	0	00111000	32768.001.00001	19	32768.002.00002	00002.02	0	10	7	2	2
10	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
11	2	2	0	10111000	32768.002.00002	0	32768.002.00002	00002.02	0	10	7	2	2
12	2	2	0	00110010	32768.003.00003	0	32768.003.00003	00003.03	0	10	7	2	2
13	2	2	0	00110010	32768.003.00003	0	32768.003.00003	00003.02	0	10	7	2	2
14	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
15	2	2	0	00111000	32768.002.00002	0	32768.002.00002	00002.02	0	10	7	2	2
16	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
17	2	2	0	00111000	32768.002.00002	0	32768.002.00002	00002.02	0	10	7	2	2
18	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
19	2	2	0	00101000	32768.001.00001	38	32768.002.00002	00002.02	0	10	7	2	2
20	2	2	0	00111000	32768.001.00001	0	32768.001.00001	00001.02	0	10	7	2	2
21	2	2	0	00101000	32768.001.00001	38	32768.002.00002	00002.02	0	10	7	2	2

Abbildung 80: Fertige Netzwerk

Wie in der obigen Abbildung zu erkennen, wird diesmal ausschließlich das Flag-Feld zur Bestimmung des BPDU-Typs verwendet. Aus diesem Grund musste beim Rapid Spanning Tree Algorithmus diese Spalte entsprechend vergrößert werden.

An den beiden vorherigen Abbildungen ist gut zu erkennen, dass das Fenster zur Anzeige der empfangenen BPDUs jetzt abhängig vom Algorithmus leicht unterschiedlich aussieht, da die Spaltenbreiten entsprechend angepasst werden müssen. [Jan10b]

11.2 Configurations

In diesem Unterkapitel wird das erweiterte Configurations-Fenster vorgestellt. An seiner grundlegenden Funktionalität und am Design hat sich im Vergleich zur Bachelorarbeit [Jan10b] kaum etwas geändert. Hauptsächlich wurde es um etliche weitere Werte ergänzt. In Abbildung 81 sind die Attribute eines beliebigen Switches von links nach rechts beim Spanning Tree Algorithmus IEEE 802.1D, beim Rapid Sapnning Tree Algorithmus und bei None zu sehen.

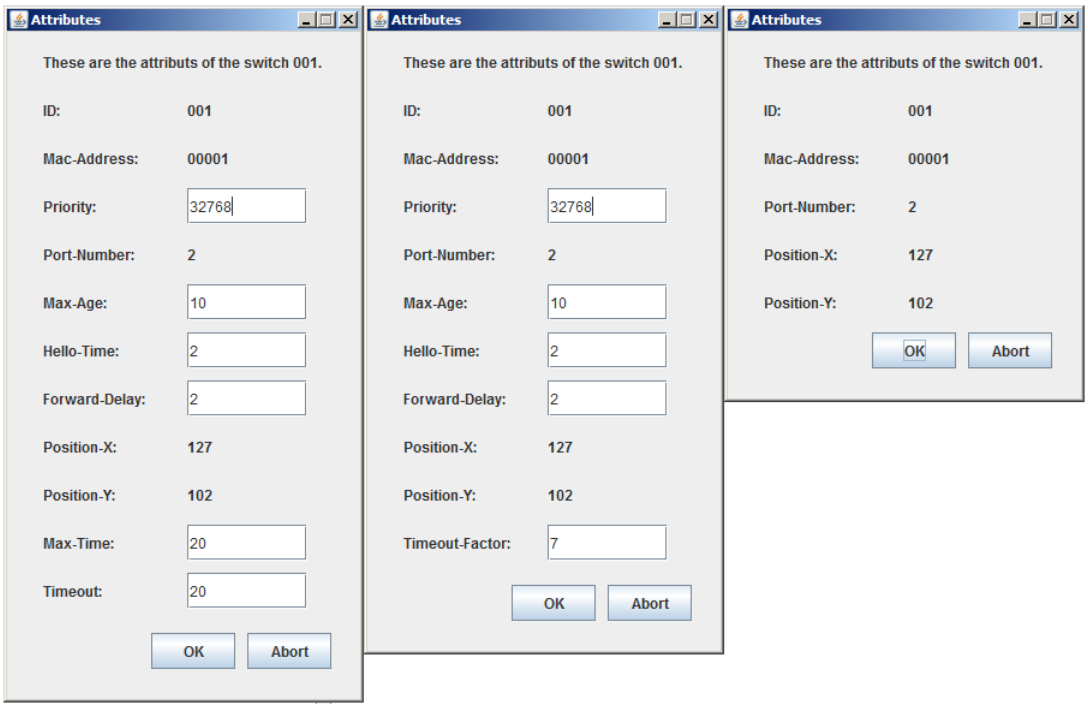


Abbildung 81: Configurations eines Switches

Attribute wie ID und Mac-Adresse sind nur durch manuelle Anpassung des Configuration-Files veränderbar. Wenn der Anwender ein Netzwerk direkt im Simulator aufbaut, hat er auf die Vergabe der Werte keinerlei Einfluss, da sich darum das Programm selbstständig kümmert. Die Positionen und die Port-Number sind darüber hinaus auch über die visuelle Darstellung im Simulator anpassbar. Diese Einschränkung wurde gemacht, damit ein angezeigtes Netz nicht mehr mittels falscher Eingaben durch den Nutzer inkonsistent und somit fehlerhaft wird. All diese Werte dienen der Darstellung und dem Aufbau eines Netzwerks.

Die restlichen Attribute können vom Nutzer bei nicht laufender Simulation nach Belieben verändert werden, da sie ausschließlich in einem der beiden Spanning Tree Algorithmen verwendet werden:

- Priority
- Max-Age

- Hello-Time
- Forward-Delay
- Max-Time
- Timeout
- Timeout-Factor

Die im Rahmen dieser Masterarbeit neu hinzugekommenen Werte sind Max-Time, Timeout und Timeout-Factor, die alle für die Rekonfiguration des Spannbaums benötigt werden. Wozu sie dienen, wurde in der Bachelorarbeit [Jan10b] und in den Kapiteln 6 und 7 erläutert.

Das Fenster, das hier besonders heraussticht ist das dritte von links, das die Attribute im Fall None zeigt. Da bei None das Netzwerk ohne jeglichen Spanning Tree Algorithmus arbeitet, werden die veränderbaren Werte nicht benötigt und deshalb auch nicht angezeigt.

Bei den Ports ist das ganze etwas einfacher. Das Fenster bei den beiden Algorithmen ist identisch und wird deshalb in der Abbildung 82 nur einmal aufgeführt.

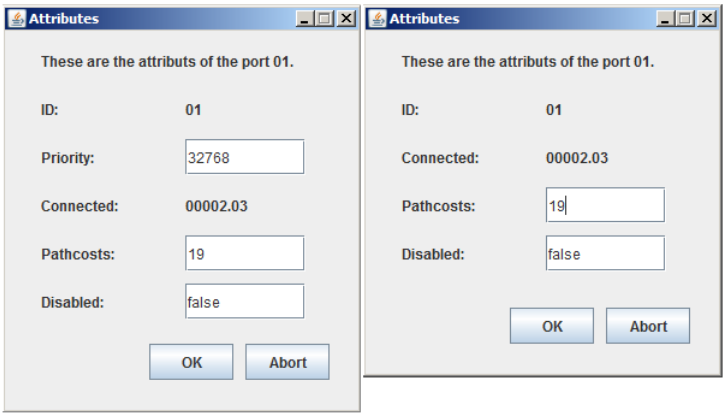


Abbildung 82: Configurations eines Ports

Das rechte Fenster zeigt wieder den Fall None. Der einzige Unterschied ist der, dass bei None keine Priorität benötigt wird. Die Pfadkosten spielen beim Verschicken von Messages eine Rolle und das Abschalten eines Ports mit Disabled ist ebenfalls sinnvoll.

Bei den Hosts ist das Fenster in allen drei Fällen gleich, da sie über keinerlei veränderbare Attribute verfügen. Dies ist in Abbildung 83 zu erkennen. [Jan10b]

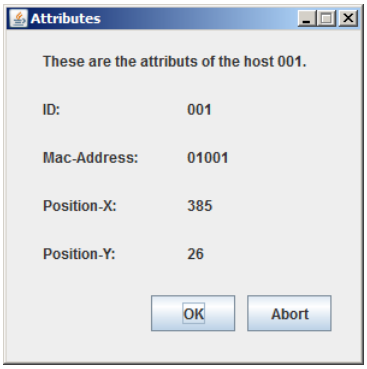


Abbildung 83: Configurations eines Hosts

11.3 Erweiterte Configuration-File

Durch die Ergänzung des Rapid Spanning Tree Algorithmus und der Rekonfiguration des Spannbaums beim Spanning Tree Algorithmus IEEE 802.1D mussten eine Reihe von neuen Attributen bei den einzelnen Netzwerk-Komponenten eingeführt werden. Die für die direkte Funktionalität der Algorithmen wichtigsten sind Max-Time, Timeout und Timeout-Factor, die in Abbildung 81 zu sehen sind. Die ersten beiden sind Bestandteil der Switches beim ursprünglichen Spanning Tree Algorithmus und das letzte gehört zum Rapid Spanning Tree Algorithmus. Alle drei stellen sozusagen den maximalen erlaubten Wert eines Timers dar. Max-Time wird für die zeitliche Alterung einer BPDU, Timeout und Timeout-Factor für das Erkennen einer ausgefallenen Verbindungsleitung verwendet. Ihre genauen Funktionsweisen wurden bereits in den Kapiteln 6 und 7 erläutert.

Um dem Nutzer eine möglichst große Flexibilität zu bieten, wurden diese drei Attribute den Configurations der Switches hinzugefügt und für ihn einstellbar gemacht. Damit diese Werte nicht bei jedem erneuten Laden desselben Configuration-Files eingegeben werden müssen, wurden die XML-Files um die drei Attribute ergänzt. Dies erforderte auch eine Anpassung des Parsers und Unparsers, da sie direkt mit diesen Files arbeiten.

Solch ein überarbeitetes Configuration-File ist in Abbildung 84 dargestellt.

```
<?xml version="1.0" ?>
<config>
<switch id="001" mac_address="00001" priority="32768" port_number="2" max_age="10" max_time="20"
    timeout_sta="20" timeout_rsta="7" hello_time="2" forward_delay="2" position_x="127" position_y="102">
<port id="00001.01" priority="32768" connected="00002.03" pathcosts="19" position="top"></port>
<port id="00001.02" priority="32768" connected="00003.03" pathcosts="19" position="bottom"></port>
</switch>

<switch id="002" mac_address="00002" priority="32768" port_number="3" max_age="10" max_time="20"
    timeout_sta="20" timeout_rsta="7" hello_time="2" forward_delay="2" position_x="289" position_y="68">
<port id="00002.01" priority="32768" connected="01001" pathcosts="19" position="right"></port>
<port id="00002.02" priority="32768" connected="00003.02" pathcosts="19" position="bottom"></port>
<port id="00002.03" priority="32768" connected="00001.01" pathcosts="19" position="left"></port>
</switch>

<switch id="003" mac_address="00003" priority="32768" port_number="3" max_age="10" max_time="20"
    timeout_sta="20" timeout_rsta="7" hello_time="2" forward_delay="2" position_x="289" position_y="168">
<port id="00003.01" priority="32768" connected="01002" pathcosts="19" position="right"></port>
<port id="00003.02" priority="32768" connected="00002.02" pathcosts="19" position="top"></port>
<port id="00003.03" priority="32768" connected="00001.02" pathcosts="19" position="left"></port>
</switch>

<host id="001" mac_address="01001" position_x="385" position_y="26"></host>
<host id="002" mac_address="01002" position_x="386" position_y="122"></host>
</config>
```

Abbildung 84: Aufbau eines Configuration-Files

Es handelt sich hier um dasselbe Netzwerk wie im Kapitel 8. Leicht zu erkennen sind die drei Switch-Tags und die zwei Host-Tags, die in Blau hervorgehoben sind. Die einfach gestalteten Hosts verfügen über genau die Attribute, die schon in Abbildung 83 zu erkennen sind. Die Switches hingegen besitzen neben ihren eigenen Attributen auch noch Kinder-Tags, mit denen die Ports realisiert wurden. Diese haben die in Abbildung 82 dargestellten Attribute.

Die drei im Rahmen dieser Masterarbeit neu hinzugekommenen Switch-Attribute heißen hier max_time, timeout_sta und timeout_rsta und befinden sich in den Switch-Tags.

Eine ausführliche Beschreibung, wie solch ein Configuration-File schrittweise aufgebaut wird, kann in der Bachelorarbeit [Jan10b] nachgelesen werden. [Jan10b]

11.4 Überarbeitung des Parsers

In diesem Unterkapitel folgt ein kurzer Blick auf den Parser, der die Configuration-Files in den Simulator einliest. Wie so ein File aufgebaut ist, kann im Unterkapitel 11.3 nachgelesen werden. An der grundlegenden Funktionsweise also dem Parsen der einzelnen Tags hat sich nichts geändert. Das größte Problem dabei stellt nach wie vor das Einlesen der Wires dar, da sie als kein separates Tag realisiert wurden und nur im Connected-Attribut der Ports vorhanden sind. Ein weiteres hieraus resultierendes Problem ist, dass die Wires geparst werden, bevor alle Ports und Host vorhanden sind. Das bedeutet, dass eine Wire bei ihrer Erzeugung eventuell noch nicht mit einem Anfang und einem Ende versehen werden kann. Deshalb müssen sie zweimal verarbeitet werden. Ein drittes Problem ist, dass beim Parsen der Ports Leitungen, die zwischen zwei Ports verlaufen, ohne entsprechende Vorkehrungen doppelt erkannt und erzeugt werden, was nicht wünschenswert ist. All diese Aspekte wurden zwar in der Bachelorarbeit [Jan10b] gelöst, aber bei der Überarbeitung im Rahmen der Masterarbeit optimiert.

Die Realisierung der Wires innerhalb der Configuration-Files wurde nicht verändert. Dies bedeutet, dass es immer noch kein separates Wire-Tag gibt und sie wie vorher auch nur durch das Connectet-Attribut der Ports definiert sind. Auch das zweimalige Bearbeiten der Wires blieb erhalten. Die eigentliche Optimierung erfolgte ausschließlich durch die bereits vorhandenen Datenstrukturen, die vieles vereinfachten.

In Listing 36 ist das Erzeugen und somit die erste Verarbeitung der Wires beim Parsen der Ports dargestellt. Es gibt beim Connected-Attribut zwei Fälle zu unterscheiden, und zwar erstens wenn es eine Verbindungsleitung zwischen zwei Ports definiert und zweitens wenn es eine Leitung zwischen einem Port und einem Host darstellt. In der ersten Situation setzt es sich aus der Mac-Adresse des Switches und der ID des Ports zusammen. Im zweiten Fall besteht es nur aus der Mac-Adresse des Hosts. Diese Prüfung liegt in der Verantwortung der ersten If-Verzweigung. Sollte es eine Wire zwischen zwei Ports sein, muss geschaut werden, ob sie nicht bereits erzeugt wurde. Deshalb wird mittels der ersten For-Schleife der Switch und mit der zweiten der entsprechende Port gesucht, der im Connected-Attribut angegeben ist. Wenn der Port gefunden wurde, bedeutet dies, dass es die Leitung bereits gibt und dass der aktuell geparste Port zum anderen Ende der Wire wird. In diesem Fall werden beide Schleifen sofort verlassen und der Parsevorgang des aktuell betrachteten Ports ist abgeschlossen.

Sollte kein entsprechender Port gefunden werden, bedeutet dies, dass es die Wire noch nicht gibt und sie erst erzeugt werden muss. Dies geschieht in der unteren If-Verzweigung, die noch vollständig abgebildet ist. Der aktuelle Port ist diesmal die erste Komponente der Wire. Da das Wire-Objekt neu erzeugt wurde, muss hier auch noch das Pathcosts-Attribut eingelesen werden.

Der Else-If-Teil, der ganz unten im Listing wegen bestehender Ähnlichkeit zum gezeigten Quellcode nur unvollständig abgebildet ist, behandelt den Fall, dass das Connected-Attribut nur die Mac-Adresse des Hosts enthält und somit eine Wire zwischen einem Host und einem Port definiert. Hier gibt es aber ein Problem, da die Host-Tags bisher noch nicht geparst wurden. Deshalb wird genauso verfahren wie gerade eben beschrieben. Es wird ein neues Wire-Objekt angelegt, der aktuelle Port wird zum Start-Port und die Pfadkosten werden eingelesen. Zum Schluss erfolgt eine Speicherung des Wire-Objekts in einer zusätzlichen HashMap, da sie nach dem Parsen der Host-Tags erneut bearbeitet werden müssen. Diese merkt sich den Port und den Inhalt des Connected-Attributs, über das das andere Ende der Leitung eindeutig identifiziert werden kann.

```
...
// Wenn connected aus zwei Teilen besteht, ist aktueller Port in jedem
// Fall mit einem anderen Port verbunden
if (parts.length == 2) {
   boolean switch_vorhanden = false;
   int port_id = new Integer(parts[1]);
   // Es wird der Switch gesucht, der mit aktuellem Port ueber Wire
   // verbunden ist
   for (String key : GUI.switch_map.keySet()) {
      // Wenn es gesuchten Switch schon gibt
      if (key.equals(parts[0])) {
         switch_vorhanden = true;
         // Suche entsprechenden Port
         for (int i = 0; i < GUI.switch_map.get(key).ports.length; i++) {
            // Wenn Port gefunden wurde, steht fest, dass aktueller
            // Port der Endport der Wire ist
            if (GUI.switch_map.get(key).ports[i].id == port_id) {
               Wire port_wire = GUI.switch_map.get(key).ports[i].wire;
               port_wire.end = aport;
               aport.wire = port_wire;
               break;
            }
         }
         break;
      }
   }
   // Wenn es den gesuchten Switch noch nicht gibt
   if (switch_vorhanden == false) {
      Wire port_wire = new Wire();
      port_wire.start = aport;
      // Lade pathcosts
      String pathcosts = parser.getAttributeValue(3);
      port_wire.path_costs = Integer.parseInt(pathcosts);
      aport.wire = port_wire;
      GUI.wire_map.put(aport, port_wire);
   }
}
// Wenn connected aus einem Teil besteht, ist aktueller Port mit einem
// Host verbunden
else if (parts.length == 1) {
...
}
```

Listing 36: Parsen der Wire-Objekte

Hiermit ist die erste Verarbeitung der Wire-Objekte abgeschlossen. Nachdem alle Tags, also auch die Host-Tags geparst wurden, müssen die Wires ein zweites Mal bearbeitet werden. Dies ist in Listing 37 zu sehen.

```
// Iteriere ueber alle Ports, die mit einem Host verbunden sind
for (Port host_port : wire_host.keySet()) {
   // Suche mittels connected verbundenen Host
   for (String key : GUI.host_map.keySet()) {
      // Wenn Host gefunden wurde
      if (GUI.host_map.get(key).mac_address.equals(wire_host.get(host_port))) {
         Host end_host = GUI.host_map.get(key);
         end_host.wire = host_port.wire;
         host_port.wire.end = end_host;
      }
   }
}
// Ueberpruefung, ob Wire auch Anfangsport und Endport hat
for (Port test_port : GUI.wire_map.keySet()) {
   if (GUI.wire_map.get(test_port).start == null || GUI.wire_map.get(test_port).end == null) {
      GUI.remove_Panel();
      ...
   }
   // Wenn es eine regulaere Wire ist
   else {
      GUI.visualize_Wire();
   }
}
```

Listing 37: Weiterverarbeitung der Wire-Objekte

Die erste For-Schleife iteriert über die HashMap, in der alle Wires gespeichert sind, die mit einem Host verbunden werden sollen. Die innere sucht in der HashMap host_map den Host, der im Connected-Attribut definiert ist. Nachdem dieser gefunden wurde, wird er zum Ende der Wire. Als Abschluss erfolgt eine Prüfung, ob nun alle Wires über einen Start-Port und eine End-Komponente verfügen. Wenn dies zutrifft, wird die Wire gezeichnet und im anderen Fall, wird eine Fehlermeldung ausgegeben und die beiden Panels geleert. [Jan10b]

11.5 Überarbeitung des Pause-Modus

Den Pause-Modus gab es zwar schon in der Bachelorarbeit [Jan10b], jedoch funktionierte er dort nur bedingt: Es dauerte einfach zu lange, bis alle Prozesse angehalten wurden und mehrmaliges Pausieren einer Simulation funktionierte oftmals überhaupt nicht.
Als im Rahmen dieser Masterarbeit noch diverse Timer hinzukamen, konnte der Pause-Modus gar nicht mehr benutzt werden. Jedoch stellt er eine wichtige Funktionalität des Spanning Tree Simulators dar. Er ermöglicht es dem Nutzer beispielsweise den Ablauf der beiden implementierten Algorithmen anzuhalten, um den aktuellen Zustand des Netzwerks untersuchen zu können. Deshalb war klar, dass er überarbeitet werden musste. In Listing 38 ist der neue Quellcode zu sehen.

```
if (anzahl_klicks == 0) {
   anzahl_klicks = 1;
}
if (anzahl_klicks == 1) {
   ...
   // Iteriere ueber alle Switches und lasse Threads warten
   for (String key : switch_map.keySet()) {
      Switch aswitch = switch_map.get(key);
      aswitch.start_pause = System.currentTimeMillis();
      aswitch.should_wait = true;
   }
   anzahl_klicks = 2;
} else if (anzahl_klicks == 2) {
   ...
   // Iteriere ueber alle Switches und stosse die Threads wieder an
   for (String key : switch_map.keySet()) {
      Switch aswitch = switch_map.get(key);
      aswitch.ende_pause = System.currentTimeMillis();
      // Passe die Timer entsprechend des pausierten Zeitraums an
      aswitch.last_send_time = aswitch.last_send_time + aswitch.ende_pause
            - aswitch.start_pause;
      aswitch.last_received_root_bpdu = aswitch.last_received_root_bpdu
            + aswitch.ende_pause - aswitch.start_pause;
      for (int i = 0; i < aswitch.ports.length; i++) {
         if (aswitch.ports[i].last_received_BPDU != 0) {
            aswitch.ports[i].last_received_BPDU = aswitch.ports[i].last_received_BPDU
                  + aswitch.ende_pause - aswitch.start_pause;
         }
         aswitch.ports[i].last_prosposal = aswitch.ports[i].last_prosposal
               + aswitch.ende_pause - aswitch.start_pause;
         aswitch.ports[i].tc_while_timer = aswitch.ports[i].tc_while_timer
               + aswitch.ende_pause - aswitch.start_pause;
      }
      aswitch.should_wait = false;
      synchronized (aswitch) {
         aswitch.notify();
      }
   }
   anzahl_klicks = 0;
}
```

Listing 38: Überarbeiteter Pause-Modus

Da das Pausieren und anschließende Starten über einen einzigen Button geregelt wird, muss bei jedem Anklicken erstmal überprüft werden, ob es das erste oder zweite Betätigen dieser Komponente ist. Im ersten Fall läuft die Simulation und soll pausiert werden. Dazu wird in der ersten For-Schleife über alle Switch-Threads iteriert und jedem durch die Variable should_wait mitgeteilt, dass er warten soll. Der eigentliche Wait-Befehl steht in der Algorithmus-Klasse, die während der laufenden Simulation verwendet wird. Dort wird regelmäßig geschaut, ob die Variable auf true gesetzt wurde. Wenn dem so ist, führt der Switch-Thread den Wait-Befehl aus. Zusätzlich zum Setzen dieser Variable wird sich auch noch der Zeitpunkt gemerkt, zudem der Befehl erteilt wurde. Diese Ergänzung wurde erst im Rahmen der Masterarbeit wichtig, da es nun eine ganze Reihe von Timern gibt, die während dem Pause-Modus unaufhaltsam weiterlaufen. Deshalb muss sich die Zeit gemerkt werden, die die Simulation pausiert.

Wenn der Nutzer den Befehl zum Weiterlaufen der Simulation erteilt, geht das Programm in den Else-If-Teil. Auch hier wird wieder über alle Switch-Threads iteriert und sich der Zeitpunkt der Button-Betätigung gemerkt. Anschließend werden alle Timer angepasst. Zum Schluss wird die should_wait-Variable zurück auf false gesetzt und alle wartenden Switch-Threads mittels notify angestoßen. [Jan10b]

12 Fazit und Ausblick

In diesem Kapitel wird beschrieben welche der geplanten Erweiterungen des Spanning Tree Simulators am Ende tatsächliche umgesetzt wurden und um welche Funktionalitäten er noch ergänzt werden könnte.

12.1 Fazit

Das Ziel dieser Masterarbeit war es den im Rahmen der Bachelorarbeit entwickelten Spanning Tree Simulator, um eine Reihe von Funktionen zu erweitern und ihn insgesamt zu verbessern. Er benötigte zu viele Systemressourcen, reagierte zu langsam und enthielt eine ganze Reihe von Fehlern.

Die erste große Erweiterung stellt die Rekonfiguration des Spannbaums beim Spanning Tree Algorithmus IEEE 802.1D dar. Sie ist zwar Bestandteil des eben genannten Algorithmus, konnte aber aus Zeitgründen in der Bachelorarbeit nicht mehr realisiert werden, was nun nachgeholt wurde.

Die zweite neue Funktionalität ist der Rapid Spanning Tree Algorithmus. Dieser stellt eine Weiterentwicklung des IEEE 802.1D dar. Er verfügt zwar über eine ähnliche grundlegende Arbeitsweise, funktioniert im Detail jedoch anders. Nun ist der Nutzer in der Lage auf ein und demselben Netzwerk abwechselnd beide Algorithmen laufen zu lassen. Diese beiden neu hinzugekommen Aspekte stellen den Hauptteil der Masterarbeit dar. Die Implementation hat jedoch deutlich länger gedauert, als ursprünglich veranschlagt wurde. Deshalb mussten einige geplante Erweiterungen weggelassen werden, die im Unterkapitel 12.2 näher erläutert werden.

Neben diesen zwei großen Erweiterungen, folgten eine ganze Reihe kleinerer, die den Simulator bereichern. Zusätzlich zur Wahl zwischen den beiden Algorithmen kann der Nutzer nun das dargestellte Netzwerk ohne einen Spanning Tree Algorithmus betreiben. Dadurch wird die Notwendigkeit eines solchen Algorithmus verdeutlicht.

Im alten Simulator konnte der Anwender Netzwerke nur mittels eines Configuration-Files darstellen lassen. Er musste das gewünschte Netz in diesem File mittels XML definieren und anschließend ins Programm laden. Dies ist sehr umständlich. Deshalb können jetzt Netzwerke direkt im Simulator über diverse Kontextmenüs erstellt werden.

Die letzte Ergänzung ist das Verschicken von Host-Nachrichten und die farbliche Markierung des von ihnen zurückgelegten Wegs.

Zusätzlich zu all diesen direkt sichtbaren Erweiterungen wurden fast alle Methoden überarbeitet, damit der Simulator weniger Systemressourcen benötigt und effizienter arbeitet. Ein Beispiel dafür ist der komplett überarbeitet Pause-Modus. Dabei wurden auch einige Fehler behoben.

Die Implementation all dieser Erweiterungen ist nur ein Bestandteil der Masterarbeit. Hinzu kommen noch die detaillierten Beschreibungen der beiden Algorithmen und ein ausführlicher Vergleich zwischen ihnen und dem Multiple Spanning Tree Algorithmus.

Alles in allem verfügt der Nutzer nun über einen gut funktionierenden Simulator, mit dem er die Arbeitsweise und die Vorteile der beiden Spanning Tree Algorithmen nachvollziehen kann. Vielleicht findet das Programm einen Einsatz in der Lehre, um den Studenten die Algorithmen näher zu bringen.

12.2 Ausblick

Wie im vorherigen Kapitel erwähnt, mussten aus Zeitgründen einige geplante Erweiterungen weggelassen werden.

Die erste ist das Einbetten des Simulators in einen Webbrowser. Dies wurde aus zweierlei Gründen nicht realisiert. Der eine war die fehlende Zeit, jedoch wurde gegen Ende der Masterarbeit auch der Sinn des Einbettens infrage gestellt. Um die implementierten Algorithmen auf einem Netzwerk laufen zu lassen, muss dieses erstmal in den Simulator gelangen. Dazu gibt es zwei Wege, und zwar wird es entweder mittels einem Configuration-File ins Programm geladen, oder es wird direkt im Simulator erstellt. Um diese jedoch in den Simulator zu laden, müssen sie von der Webseite heruntergeladen werden. Da der Nutzer jetzt eh bestimmte Dateien downloaden muss, liegt es nahe auch die ausführbare Jar zum Download anzubieten.

Der zweite Aspekt, der nicht umgesetzt werden konnte, ist die Zeitmessung. Der Anwender sollte die Möglichkeit bekommen, die Zeit zu messen, die die beiden Algorithmen zum Aufbau des Spannbaums benötigen. Es hat sich aber herausgestellt, dass dies nicht so einfach zu realisieren ist, da der Simulator keine Möglichkeit hat, zu bestimmen, ob der Spannbaum nun fertig ist oder nicht. Dieses Problem besteht, weil es keine übergeordnete Instanz gibt, die das gesamte Netzwerk kennt und den Algorithmus steuert. Aber genau diese Dezentralisierung ist einer der zentralen Aspekte der Spanning Tree Algorithmen. Für die Umsetzung dieser Erweiterung wurden sich drei verschiedene Realisierungen überlegt, die im Folgenden für zukünftige Entwickler erläutert werden.

1. Die erste einfache und relativ sichere Methode ist die, sich jeden Switch den Zeitpunkt des Eintreffens einer neuen BPDU in einer Variablen merken zu lassen. Bei jeder weiteren neuen Nachricht wird der alte Wert der Variablen überschrieben. Sollte der Switch die dritte identische BPDU erhalten, weiß er, dass er fertig ist und setzt dazu eine entsprechende boolsche Variable. Sobald alle Switches diese Variable gesetzt haben, wird der größte der gemerkten Zeitwerte als Dauer für den Aufbau des Spannbaums verwendet. Sollten drei identische Nachrichten nicht ausreichen, kann die Anzahl nach Belieben erhöht werden.

2. Die zweite Möglichkeit ist die, kontinuierlich alle Leitungen, die mit zwei Ports verbunden sind, zu überprüfen, ob bereits alle nur noch über genau einen Designated-Port verfügen. Im fertigen Spannbaum darf jede Leitung, die zwei Ports miteinander verbindet und nicht deaktiviert wurde, immer nur genau einen Designated-Port besitzen. Auf diese Weise könnte die Zeit theoretisch gemessen werden. Die genannte Bedingung trifft zwar im fertigen Spannbaum zu, aber ob der getestete Zustand vielleicht auch kurzfristig beim Aufbauprozess entsteht, ist ohne ausreichende Überprüfungen nicht sicher.

3. Bei der dritten und sichersten Methode, ist der Anwender selbst gefordert. Dazu lässt er einen der beiden Algorithmen auf dem dargestellten Netzwerk laufen und wartet solange,

bis er sicher ist, dass der Spannbaum fertig aufgebaut wurde. Dann gibt er dem Simulator beispielsweise mittels eines Buttons den Befehl sich das Netzwerk mit allen Einstellungen zu merken. Damit ist gemeint, dass alle Komponenten mit allen Attributen wie zum Beispiel die Portart gespeichert werden. Nachdem sich der Simulator das Netz gemerkt hat, muss der Anwender den Algorithmus nochmals laufen lassen. Aber diesmal verfügt der Simulator über eine Referenz, in der der fertige Spannbaum bereits definiert ist. Mittels kontinuierlichen Vergleichen zwischen dem Referenzbaum und dem aktuellen Spannbaum, kann das Ende des Aufbaus ziemlich genau bestimmt werden.

Nun folgt eine Liste mit weiteren denkbaren Erweiterungen:

- Mehr als vier Ports pro Switch
- Verwaltung der Threads
- Weitere Spanning Tree Algorithmen
- Zoomen im dargestellten Netzwerk
- Geschwindigkeit der Simulation steuern

Mit der Verwaltung der Threads ist gemeint, dass es sinnvoll wäre eine Java-Klasse zu schreiben, die sich um die Steuerung der Switch-Threads kümmert. Bis jetzt tut dies ausschließlich die JVM. Diese gewährleistet jedoch keine Fairness in der Ausführung der Threads, weshalb einige länger warten müssen als andere.

13 Anhang

In diesem letzten Kapitel werden zuerst die an den Spanning Tree Algorithmus gestellten Anforderungen und anschließend alle Dokumente und Dateien aufgelistet, die auf der dieser Masterarbeit beigefügten CD enthalten sind.

13.1 Anforderungen

In diesem Kapitel sind alle Anforderungen bezüglich des Spanning Tree Simulators erfasst und nummeriert aufgelistet. Es gibt neben den Systemgrenzen zwei Arten von Anforderungen, und zwar die funktionalen und die nicht funktionalen. Die letzte Art umfasst Aspekte, die beispielsweise das Aussehen der GUI oder den Programmierstil betreffen und die funktionalen Anforderungen beziehen sich ausschließlich auf die reine Funktionsweise des Programms. Neben dieser Unterteilung gibt es noch eine weitere, und zwar nach den drei folgenden Kriterien:

- Pflicht
- Soll
- Kann

Wie die Namen schon verraten, legen diese drei Kriterien eine Rangfolge der Wichtigkeit fest. Diese nimmt vom ersten bis zum letzten Punkt immer weiter ab. Diese Kennzeichnung befindet sich in eckigen Klammern am Ende jeder Anforderung des funktionalen und nicht funktionalen Unterkapitels.
Da diese Arbeit auf die Bachelorarbeit [Jan10b] aufbaut, werden die bereits dort aufgeführten Anforderungen nur in Ausnahmefällen erneut aufgelistet.

13.1.1 Systemgrenzen

Die Systemgrenzen definieren den groben Rahmen des in dieser Masterarbeit erweiterten Spanning Tree Simulators.

1. Als Zielplattform wird Windows Vista (32 und 64 Bit) und Windows 7 (32 und 64 Bit) angestrebt.
2. Es kann immer nur ein Netzwerk zur selben Zeit im Simulator angezeigt werden.
3. Die implementieren Algorithmen können immer nur auf dem angezeigten Netzwerk laufen gelassen werden.
4. Es kann immer nur ein Algorithmus zu selben Zeit auf dem dargestellten Netzwerk laufen.
5. Der „Multiple Spanning Tree Algorithmus“ ist nicht Bestandteil des Spanning Tree Simu-

lators.

6. Da der Spanning Tree Simulator für die Lehre entwickelt wurde, ist die Größe von verwendbaren Netzwerken eher als klein einzustufen.

13.1.2 Funktionale Anforderungen

In diesem Unterkapitel werden alle funktionalen Anforderungen aufgelistet.

Deaktivieren/Aktivieren von Leitungen

7. Der Nutzer muss nach dem erfolgreichen Aufbau des Spannbaums in der Lage sein, einzelne Leitungen des Netzwerks zu deaktivieren. [Pflicht]

8. Deaktivierte Leitungen müssen kenntlich gemacht werden. [Pflicht]

9. Mithilfe des Kontextmenü-Punkts „Deactivate Port“ eines Ports soll die angeschlossene Leitung deaktiviert werden. [Soll]

10. Der Menü-Punkt „Deactivate Port“ sollte nur sichtbar sein, wenn die am angeklickten Port angeschlossene Leitung aktiviert ist. [Soll]

11. Über eine deaktivierte Leitung dürfen keinerlei Nachrichten ausgetauscht werden. [Pflicht]

12. Der Nutzer muss in der Lage sein, deaktivierte Leitungen wieder zu aktivieren. [Pflicht]

13. Ein deaktivierte Leitung soll mittels des Kontextmenü-Punkts „Activate Port“ eines an sie angeschlossenen Ports wieder aktiviert werden können. [Soll]

14. Dieser Menü-Punkt soll nur sichtbar sein, wenn die am angeklickten Port angeschlossene Leitung deaktiviert ist. [Soll]

15. Mittels des Kontextmenü-Punkts „Deactivate all Ports of Switch“ soll der Nutzer alle Leitungen eines Switches gleichzeitig deaktivieren können. [Soll]

16. Über den Kontextmenü-Punkt „Activate all Ports of Switch“ soll der Nutzer alle deaktivierten Leitungen eines Switches gleichzeitig aktivieren können. [Soll]

17. All diese Menü-Punkte sollen nur während einer laufenden Simulation sichtbar sein. [Soll]

Rekonfiguration des Spanning Trees beim STA

18. Der BPDU-Typ „Topology-Change-Notification-BPDU“ muss implementiert werden. [Pflicht]

19. Alle BPDUs benötigen eine zeitliche Alterung und ein maximal erlaubtes Alter.[Pflicht]

20. Nach Ablauf der Lebensspanne einer BPDU muss diese gelöscht werden. [Pflicht]

21. Jeder Switch muss über ein Timeout-Intervall verfügen. [Pflicht]

22. Eine Timeout-Situation muss eintreten, wenn ein Switch bis zum Ende des Intervalls über seinen Root-Port keine BPDU vom Root-Switch empfängt. [Pflicht]

23. Wenn ein Timeout eingetreten ist, muss der betroffene Switch über all seine Ports eine TCN-BPDU verschicken. [Pflicht]

24. Der betroffene Switch muss nun auf die Antwort vom Root-Switch warten. [Pflicht]

25. Wenn ein Switch eine TCN-BPDU empfängt, schickt er solch eine BPDU über seinen Root-Port zum Root-Switch und vergisst die Root-ID und die Root-Pathcosts. [Pflicht]

26. Danach verfährt er genauso wie der Switch, bei dem das Timeout eingetreten ist. [Pflicht]

27. Wenn der Root-Switch eine TCN-BPDU empfängt, muss er eine Antwort-BPDU verschicken. [Pflicht]

28. Sobald ein Switch, der auf eine Antwort-BPDU wartet, solch eine empfängt, wertet er diese aus und schickt sie über seine Ports weiter. [Pflicht]

29. Wenn ein Switch die Antwort-BPDU nicht rechtzeitig empfängt, geht er davon aus, dass er selbst der Root-Switch ist. [Pflicht]

30. Der Nutzer muss Host-Nachrichten verschicken können. [Pflicht]

31. Host-Nachrichten dürfen nur von Ports empfangen und weitergeleitet werden, die sich im Forwarding-Status befinden und keine Blocking-Ports sind. [Pflicht]

Rapid Spanning Tree Algorithmus

32. Der bisherige Aufbau einer BPDU muss an den Rapid Spanning Tree Algorithmus angepasst werden. [Pflicht]

33. Es muss die folgenden Portarten geben: [Pflicht]

    - Root-Port
    - Designated-Port
    - Alternate-Port
    - Backup-Port
    - Edge-Port

34. Über Alternate- und Backup-Ports dürfen keine Nachrichten verschickt werden. [Pflicht]

35. Über Edge-Ports dürfen keine BPDUs verschickt werden. [Pflicht]

36. Es muss die folgenden Portstatus geben: [Pflicht]

    - Disabled
    - Discarding
    - Learning
    - Forwarding

37. Ports, die sich im Disabled-Status befinden, dürfen keinerlei Nachrichten empfangen oder verschicken. [Pflicht]

38. Nur Ports im Forwarding-Status dürfen Host-Nachrichten empfangen und verschicken. [Pflicht]

39. Alle Switches gehen zu Beginn davon aus, dass sie der Root-Switch sind. [Pflicht]

40. Alle Ports sind zu Beginn Designated-Ports und befinden sich im Discarding-Status. [Pflicht]

41. Jeder Switch verschickt über all seine Ports, die sich in dem eben genannten Zustand befinden, Proposal-BPDUs an all seine Nachbar-Switches. [Pflicht]

42. Danach warten die entsprechenden Ports auf eine Agreement-BPDU. [Pflicht]

43. Wenn ein Switch eine Proposal-BPDU empfängt, wertet er sie aus und schickt eine Agreement-BPDU zurück.

44. Wenn ein Switch eine Agreement-BPDU empfängt, setzt er den Empfangsport sofort in den Forwarding-Status.

45. Wenn ein Switch keine Agreement-BPDU empfängt, geht er, wie in der Bachelorarbeit [Jan10b] beschrieben, mithilfe des Forward-Delay-Timers in den Forwarding-Status. [Pflicht]

46. Jeder Switch verschickt nach Ablauf der Hello-Time über all seine Root- und Designated-Ports normale BPDUs. [Pflicht]

47. Jeder Port muss über ein eigenes Timeout-Intervall verfügen. [Pflicht]

48. Wenn ein Switch über seinen Root-Port nicht innerhalb des Intervalls eine BPDU von seinem Nachbar-Switch empfängt, muss ein Timeout eintreten. [Pflicht]

49. Dann verschickt er über all seine Ports eine TC-BPDU und startet den TC-While-Timer. [Pflicht]

50. Nach Ablauf dieses Timers geht der Switch davon aus, dass er selbst der Root-Switch ist. [Pflicht]

51. Wenn ein Switch eine TC-BPDU empfängt, schickt er eine TC-BPDU über alle Designated-

und Root-Ports. [Pflicht]

52. Danach startet der Switch ebenfalls seinen TC-While-Timer. [Pflicht]

53. Der Nutzer muss Host-Nachrichten verschicken können. [Pflicht]

Abschalten der Spanning Tree Algorithmen

54. Der Nutzer muss in der Lage sein, das dargestellte Netzwerk ohne einen Spanning Tree Algorithmus zu verwenden. [Soll]

55. Der Nutzer muss Host-Nachrichten verschicken können. [Soll]

Verschicken von Host-Nachrichten

56. Es muss möglich sein, während einer laufenden Simulation Nachrichten von einem Host zu einem anderen zu verschicken. [Pflicht]

57. Über den Kontextmenü-Punkt „Send Message to“ eines Hosts, sollte eine Nachricht verschickt werden können. [Soll]

58. Ein Host sollte keine Nachricht an sich selbst schicken können. [Soll]

59. Die Switches müssen mittels einer SAT-MAC-Table lernen, über welchen Port ein bestimmter Host zu erreichen ist. [Pflicht]

60. Diese Nachrichten brauchen keinen Inhalt zu haben.[Kann]

Visualisierung des Wegs einer Host-Nachricht

61. Der Weg, den eine Host-Nachricht nimmt, muss graphisch hervorgehoben werden. [Pflicht]

62. Die Leitungen, auf denen sich eine Host-Nachricht gerade befindet, müssen kenntlich gemacht werden. [Pflicht]

63. Die Leitungen, die bereits wieder von einer Host-Nachricht verlassen wurden, müssen markiert werden. [Pflicht]

64. Bei jedem Verschicken einer Host-Nachricht müssen zuerst alle Markierungen, die den Weg einer vorherigen Nachricht kennzeichnen, entfernt werden. [Pflicht]

Erstellen eines Netzwerks im Simulator

65. Der Nutzer muss in der Lage, sein direkt im Simulator ein beliebiges Netzwerk bestehend aus Switches, Ports, Hosts und Wires zu erstellen. [Pflicht]

66. Der Anwender sollte über den Kontextmenü-Punkt „Create Switch“ des Panels zur Darstellung eines Netzwerks Switch-Komponenten erzeugen können. [Soll]

67. Über den Kontextmenü-Punkt „Create Host“ des Panels zur Darstellung eines Netzwerks sollen Hosts erzeugt werden. [Soll]

68. Mittels des Kontextmenü-Punkts „Create Port“ eines Switches sollen bis zu vier Ports pro Switch erzeugt werden. [Soll]

69. Mithilfe des Kontextmenü-Punkts „Connect to Switch“ eines Ports soll eine Leitung zu einem anderen Switch kreiert werden. [Soll]

70. Der Kontextmenü-Punkt „Connect to Host“ eines Ports soll eine Leitung zu einem Host erschaffen. [Soll]

71. Diese Menü-Punkte dürfen nur vor dem Starten oder nach dem Beenden einer Simulation sichtbar sein. [Pflicht]

72. Die Attribute der jeweiligen Objekte müssen mit den in einem Configuration-File stehenden Werten initialisiert werden. [Pflicht]

73. Jeder Port und jeder Host darf nur mit einer Leitung verbunden sein. [Pflicht]

74. Eine Simulation darf nur gestartet werden, wenn folgende Kriterien erfüllt sind: [Pflicht]

    - Jeder Switch muss mindestens einen Port haben.
    - Jeder Port muss mit genau einer Leitung verbunden sein.
    - Jeder Host muss mit genau einer Leitung verbunden sein.

75. Mithilfe des Kontextmenü-Punkts „New Network“ des Panels zur Darstellung eines Netzwerks soll der Inhalt dieses Panels mit allen dazugehörigen Objekten vollständig gelöscht werden. [Soll]

76. Es muss möglich sein jede einzelne Komponente des Netzwerks also jeden Switch, Port und Host einzeln zu entfernen. [Pflicht]

77. Über die Kontextmenü-Punkte „Delete Switch“, „Delete Port“ und „Delete Host“ der jeweiligen Komponenten, sollen diese gelöscht werden. [Soll]

78. Wenn eine dieser drei Komponenten gelöscht wird, muss die angeschlossene Wire, falls diese vorhanden ist, ebenfalls entfernt werden. [Pflicht]

79. Nach dem Löschen einer Komponente muss erneut geprüft werden, ob die drei Kriterien zum Starten einer Simulation noch erfüllt sind. [Pflicht]

Anzeige der empfangenen Configuration-Messages

80. Es müssen auch die neuen BPDU-Typen angezeigt werden können. [Pflicht]

81. Das Fenster zur Anzeige von empfangenen Configuration-Messages soll auch während der laufenden Simulation aufrufbar sein. [Soll]

82. Die Größe des Fensters soll veränderbar sein. [Soll]

83. Es soll möglich sein, mehrere dieser Fenster gleichzeitig anzeigen zu lassen. [Soll]

13.1.3 Nicht funktionale Anforderungen

In diesem Unterkapitel befinden sich alle nicht funktionalen Anforderungen, also solche, die sich beispielsweise auf die Darstellung des Simulators beziehen.

Auswahl des Algorithmus

84. Der Nutzer soll in der Lage sein, über die drei Radio-Buttons „Spanning Tree Algorithmus“, „Rapid Spanning Tree Algorithmus“ und „None“ zu entscheiden, ob er auf dem dargestellten Netzwerk den Spanning Tree Algorithmus IEEE 802.1D, den Rapid Spanning Tree Algorithmus oder keinen der beiden laufen lassen möchte. [Soll]

85. Bei den beiden Algorithmen kann die Rekonfiguration des Spannbaums über die Checkbox „Activate reconfiguration of the spanning tree“ an- bzw. abgeschaltet werden. [Kann]

86. Wenn der Radio-Button None gewählt wurde, kann die Alterung der SAT-MAC-Table über die Checkbox „Activate aging of SAT-MAC-Table entries.“ an- bzw. abgeschaltet werden. [Kann]

Farbliche Markierung der Wires

87. Deaktivierte Leitungen sollen rot eingefärbt werden. [Soll]

88. Die Leitung, auf der sich gerade eine Host-Nachricht befindet, soll grün dargestellt werden. [Soll]

89. Die Leitung, die von einer Host-Nachricht schon wieder verlassen wurde, soll orange dargestellt werden. [Soll]

Anzeige der empfangenen Configuration-Messages

90. Das Fenster zur Anzeige der empfangenen Configuration-Messages soll mit einer JTable realisiert werden. [Soll]

Restliche Anforderungen

91. Die Benutzeroberfläche des Spanning Tree Simulators soll in englischer Sprache geschrieben sein. [Soll]

92. Der Quelltext muss kommentiert werden. [Soll]

93. Die Methoden und Klassen müssen mit JavaDoc dokumentiert werden. [Pflicht]

94. Die Javakonventionen von Oracle zur Entwicklung von Programmen sollten eingehalten werden. [Soll]

95. Der Simulator muss erfolgreich getestet werden. [Pflicht]

96. Es muss eine ausführbare Jar-Datei erstellt werden. [Pflicht]

13.2 Inhalt der CD

Die folgenden Dateien befinden sich auf der beigefügten CD:

- Vollständiger Quellcode des Spanning Tree Simulators
- Ausführbare Jar-Datei
- Einige Configuration-Files
- Eclipse-Export des gesamten Projekts
- Gesamte JavaDoc
- Dokumentation der Masterarbeit

Abbildungsverzeichnis

Literatur

[Ava10] Avaya: RSTP/MSTP Technical Configuration Guide. http://support.avaya.com/css/P8/documents/100123883. Version: 7 2010

[Bad05] Badach, Anatol: Protokolle und Dienste der Informationstechnologie. Heinz Schulte WEKA-Verlag, 2005. – 3600 S. http://www.competence-site.de/downloads/54/b7/i_file_335429/competence_site_badach_rstp.pdf

[Cis06] Cisco: Understanding Rapid Spanning Tree Protocol (802.1w). http://www.cisco.com/en/US/tech/tk389/tk621/technologies_white_paper09186a0080094cfa.shtml. Version: 10 2006

[eTu12] eTutorials.org: Introducing Rapid Spanning Tree Protocol. http://etutorials.org/Networking/Lan+switching+fundamentals/Chapter+10.+Implementing+and+Tuning+Spanning+Tree/Introducing+Rapid+Spanning+Tree+Protocol/. Version: 2008-2012

[Hei07] 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: 5 2007

[Hei09] 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: 7 2009

[Jan10a] Janke, Andreas S.: Handhabung des Programms, 3 2010

[Jan10b] Janke, Andreas S.: Implementierung des Spanning Tree Algorithmus IEEE 802.1D, Universität Koblenz - Landau Abteilung Koblenz, Bachelorarbeit, 3 2010

[Lap08] Lapukhov, Petr: MSTP Tutorial Part I: Inside a Region. http://blog.ine.com/2008/07/27/mstp-tutorial-part-i-inside-a-region/. Version: 2008

[Lip] Lipinski, Klaus: MSTP (multiple spanning tree protocol). http://www.itwissen.info/definition/lexikon/MSTP-multiple-spanning-tree-protocol.html

[Ras08] Raschke, Patrick: Aufbau einer experimentellen Spanning Tree Umgebung. https://wiki.netlab.inf.hochschule-bonn-rhein-sieg.de/index.php/Praxissemesterprojekt_Spanning_Tree. Version: 12 2008

[Sch03] Schulte, Wolfgang: Spanning Tree. http://www.funkschau.info/heftarchiv/pdf/2003/fs1603/fs0316055.pdf. Version: 2003

[Wik11] Rapid Spanning Tree Protocol. http://de.wikipedia.org/wiki/Rapid_Spanning_Tree_Protocol. Version: 7 2011