Universität Koblenz-Landau
Fachbereich Informatik
Institut für Informatik
Arbeitsgruppe Rechnernetze

Performanz von RIP-MTI

Diplomarbeit
zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Dean Wickert

Betreuer: Prof. Dr. C. Steigner, Dipl. Inform. H. Dickel
Erstgutachter: Prof. Dr. C. Steigner
Zweitgutachter: Dipl. Inform. H. Dickel

Koblenz im März 2009

Erklärung

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

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

Koblenz, den 9. März 2009

.....

Dean Wickert

Inhaltsverzeichnis

1 Einleitung

RIP, das Routing Information Protocol und Grundlage des später entwickelten RIP-MTI, gehört zur Familie des Distanz-Vektor-Routings. Dieses wurde bereits in der Mitte des 20. Jahrhunderts im ARPANET eingesetzt. Bekannter Vertreter des Verfahrens neben RIP ist IGRP (Interior Gateway Routing Protocol). Beide Routingprotokolle haben die Schwäche des Distanz-Vektor-Routings geerbt und bis heute damit zu kämpfen. Gemeint ist das Counting to infinity Problem.

Für RIP hat A. Schmidt 1999 in seiner Diplomarbeit mit der Entwicklung des "Routing Information Protocol with Minimum Topology Information" (RIP-MTI) eine Möglichkeit vorgestellt das Counting to infinity Problem erfolgreich zu bekämpfen [SCH99]. RIP-MTI bietet mit der Erkennung topologischer Schleifen einen vielversprechenden Ansatz, um Routingschleifen zu verhindern und somit die Konvergenzzeit in RIP zu verbessern.

Diese Diplomarbeit beschäftigt sich mit der Performanz von RIP-MTI, insbesondere mit der Performanz der Schleifenerkennung. Ziel der Arbeit ist es die Zeitdauer der Schleifenerkennung von RIP-MTI zu untersuchen und Probleme, welche bei der Erkennung von Schleifen auftreten können, aufzudecken und zu lösen. In Kapitel 2 werden zunächst die benötigten Grundlagen des Routing Information Protocol (RIP) beschrieben. Kapitel 3 beschäftigt sich mit der Performanz von RIP-MTI. Die Zeitdauer der Schleifenerkennung (Kapitel 3.1) sowie Probleme bei der Schleifenerkennung (Kapitel 3.2) werden genauer untersucht. Die dabei entdeckte Möglichkeit zur Verbesserung der Schleifenerkennung nach dem Start von RIP-MTI wurde Slow Start genannt und wird in Kapitel 3.3 beschrieben. Nebenbei ergibt sich durch Slow Start noch die Möglichkeit beim Start des Routers (egal, ob es sich um einen klassischen RIP-Router oder um einen RIP-MTI-Router handelt) mehr als 50% des durch den Start verursachten Nachrichtenverkehrs im gesamten Netzwerk einzusparen. Die Validierung mit VNUML (Virtual Network User Mode Linux) zeigt in Kapitel 4, wie es zu diesem Einsparungsverhältnis kommt.

2 Routing Information Protocol (RIP)

Das folgende Kapitel beschreibt den Aufbau und die Entwicklung des Routing Information Protocol (RIP) bis hin zu der Erweiterung „Routing Information Protocol with Minimum Topology Information“, später auch als „Routing Information Protocol with Metric based Topology Investigation“ (RIP-MTI) bezeichnet.

Bei RIP handelt es sich um einen dynamischen Routingalgorithmus, bei dem jeder Router seine Routingtabelle selbst erstellt. Dynamische Routingalgorithmen wurden notwendig, da mit wachsender Größe des ARPANETs[1] statische Routen, also feste Eintragungen in die Routingtabelle durch den Administrator, zu zeitaufwendig und arbeitsintensiv wurden. Der Administrator musste beispielsweise ständig Aktualisierungen vornehmen, wenn ein Netzwerk hinzugefügt oder entfernt wurde, bei Ausfällen von Netzwerken musste eine alternative Route eingetragen werden.

Deshalb entwickelten Bellman, Ford und Fulkerson das Bellman-Ford-Routing [TAN04]. Bezeichnungen sowie Jahreszahlen unterscheiden sich je nach Literatur.[2]

Der dynamische Routingalgorithmus von Bellman, Ford und Fulkerson gehört zur Familie des Distanz-Vektor-Routings ([TAN04], Seite 395ff) und war entsprechend einfach gehalten. Jeder Router musste nur die folgenden drei Regeln befolgen:

1. Nimm alle direkt angeschlossenen Netzwerke mit der Entfernung 1 in deine Routingtabelle auf.
2. Teile deine Routingtabelle regelmäßig deinen Nachbarroutern mit.
3. Erhältst du von deinem Nachbarrouter Informationen zu einem Netzwerk, welches nicht oder mit einer um mehr als 1 größeren Entfernung in deiner Routingtabelle enthalten ist, so trage dieses Netzwerk mit einer um 1 erhöhten Entfernung in deine eigene Routingtabelle ein.

Da dieser Routingalgorithmus auf einem Suchalgorithmus (Suche nach dem kürzesten Weg) beruht, könnte statt dem in den drei Regeln genannten einheitlichen Wert von 1 auch ein Wert entsprechend der tatsächlichen Kosten (z.B. Netzwerkgeschwindigkeit, anfallende Trafficgebühren) für die einzelnen Teilstrecken / -netze eingesetzt werden. Wichtig für die Funktionsweise des Algorithmus ist nur, dass die Entfernung zwischen

1 ARPANET: Advanced Research Projects Agency Network, ein frühes amerikanisches Computernetzwerk. [TAN04] gibt auf Seite 67 die Entstehung des ARPANETs gegen Ende der 1950er Jahre an. Laut [WIKI03] wurde das ARPANET 1962 in Auftrag gegeben.

2 Das Bellman-Ford-Routing wird in [TAN04] auf Seite 395 auch als Ford-Fulkerson-Algorithmus bezeichnet. [WIKI02] spricht vom Bellman-Ford-Algorithmus sowie vom Moore-Bellman-Ford-Algorithmus. Als Entwicklungszeitraum für diesen Algorithmus – sofern überhaupt angegeben – finden sich Jahresangaben von 1956 bis 1962.

zwei Routern nirgends kleiner oder gleich null ist. In diesem Fall kann der Algorithmus versagen.

Wegen seiner Einfachheit entwickelte sich das Bellman-Ford-Routing zu **dem** Routingalgorithmus des ARPANETs. Später wurde der Algorithmus dann im Internet unter der Bezeichnung RIP (Routing Information Protocol) eingesetzt.

Mittlerweile gibt es verschiedene Versionen von RIP, welche nachfolgend beschrieben werden.

2.1 RIPv1

Die erste Version des RIP wird im RFC 1058 [RFC1058] vom Juni 1988 beschrieben und ist als „historisch“ gekennzeichnet. Die folgend beschriebenen Grundprinzipien finden sich jedoch weitgehend auch bei den neueren RIP-Versionen. In den nächsten Kapiteln werden deshalb nur noch die Änderungen zur jeweils vorherigen Version dargestellt.

2.1.1 Funktionsweise

Bei RIP handelt es sich wie auf der vorherigen Seite beschrieben um ein sehr einfaches Routingprotokoll. Jeder Router arbeitet nach den dort grob skizzierten drei Regeln. Dies funktioniert folgendermaßen:
Beim Einschalten erkennt der Router, an welche Netzwerke er direkt angeschlossen ist. Diese Netzwerke werden nach Regel 1 mit der Entfernung (Metrik) 1 in seine Routingtabelle eingetragen. Die Einträge der Routingtabelle setzen sich zusammen aus:

- der Nummer des Netzwerkes.
- der Metrik.
- dem Anschluss (bzw. Nachbarrouter), über welchen dieses Netzwerk erreichbar ist.

Jeder Router teilt nun die Informationen aus seiner Routingtabelle in mehr oder weniger regelmäßigen Abständen durch die sogenannten Timed Updates seinen Nachbarroutern mit, wie es Regel 2 fordert. Prinzipiell erfolgt alle 30 Sekunden ein Timed Update. Allerdings ist der zugehörige Timer bei jedem Intervall mit einem zufälligen Offset von ± 0 bis 5 Sekunden versehen, so dass zwischen zwei Timed Updates eine Zeitspanne von 25 bis 35 Sekunden liegen kann. Dies wurde eingeführt als sich gezeigt hat, dass sich die einzelnen Router mit der Zeit synchronisieren und somit die Update-Nachrichten gleichzeitig verschickt werden [FLO94]. Der zufällige Offset verhindert eine Synchronisation und die durch die Update-Nachrichten entstehende Netzlast wird zeitlich gleichmäßiger verteilt.

Abbildung 2.1 zeigt den Aufbau einer solchen Nachricht. Ihr ist ein Header vorangestellt, in welchem die Version von RIP (hier immer 1) sowie das auszuführende Kommando angegeben wird. Es gibt zwei verschiedene Kommandos. Das Response- oder auch Update-Kommando übermittelt mit den Routeneinträgen die Informationen der eigenen Routingtabelle (Protokolltyp, Adresse, Metrik) an die Nachbarrouter. Eine Update-Nachricht kann bis zu 25 Routeneinträge enthalten, welche jeweils aus dem Adress- bzw. Protokolltyp, der Netzwerkadresse, der Metrik sowie einigen Platzhaltern (Nullbytes) bestehen.
Es gibt jedoch nicht nur Update-Nachrichten, sondern auch Request-Nachrichten. Das Request-Kommando dient dem Router dazu gezielt Routeneinträge seiner Nachbarrouter abzufragen. Es wird zum einen beim Einschalten des Routers genutzt, damit dieser Router direkt die Routingtabellen seiner Nachbarrouter erfragen kann. Zum anderen kann das Request-Kommando genutzt werden, um für Diagnosezwecke gezielt Informationen zu einzelnen Netzwerken abzurufen.

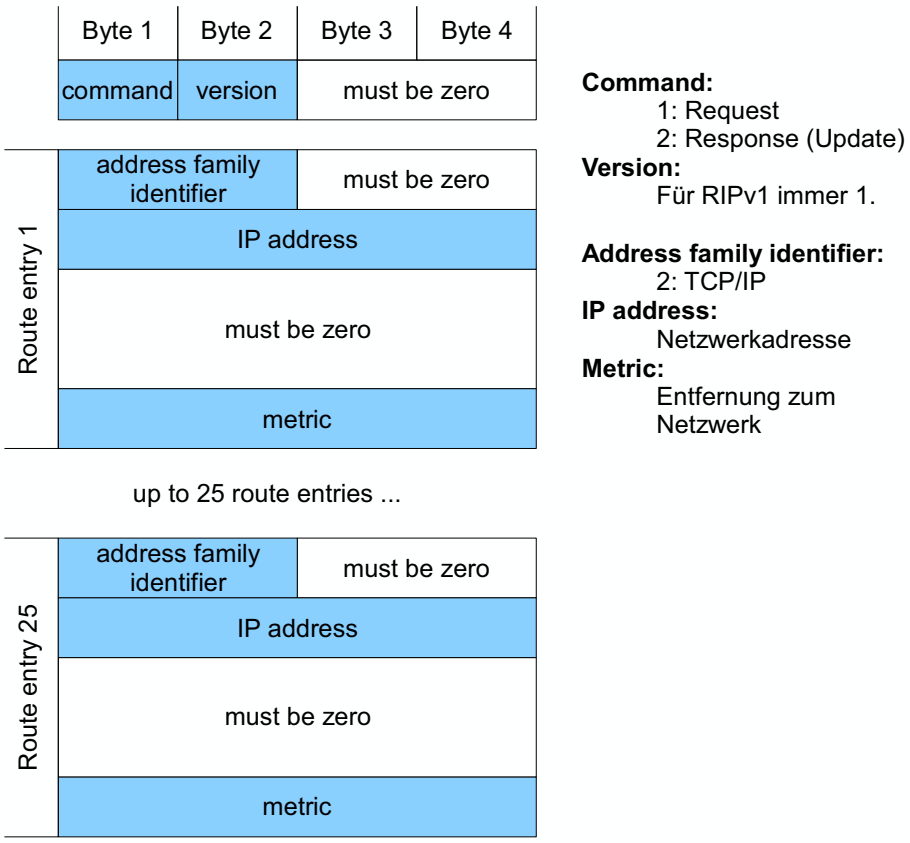


Abbildung 2.1: Nachrichtenformat RIP Version 1

Erhält ein Router nun eine solche, eben beschriebene Update-Nachricht von seinem Nachbarrouter, so prüft er gemäß Regel 3, ob diese Routeneinträge zu ihm unbekannten

Netzwerken enthält. Ist das der Fall, wird der entsprechende Routeneintrag in die eigene Routingtabelle übernommen und die übermittelte Metrik dabei um 1 erhöht.
Weiterhin wird überprüft, ob die Update-Nachricht Routeneinträge enthält, deren Metrik mehr als eins kleiner ist als die zu diesem Netzwerk gespeicherte Metrik seiner eigenen Routingtabelle. In diesem Fall existiert ein kürzerer Pfad als der gespeicherte zu dem jeweiligen Netzwerk. Der Router aktualisiert dann den aktuell vorhandenen Eintrag in seiner Routingtabelle. Die vom Nachbarrouter übermittelte Metrik wird hierzu um 1 erhöht und bei dem betreffenden Netzwerk in der Routingtabelle gespeichert. Außerdem wird der Absender der Update-Nachricht bei den Erreichbarkeitsinformationen zu diesem Netzwerk eingetragen. Die alten Metrik- und Erreichbarkeitsinformationen gehen verloren.

Bei Ausfällen von Teilnetzen kann es passieren, dass sich der Pfad zu einem bestimmten Netzwerk verlängert. Deshalb muss der Router bei eingehenden Update-Nachrichten auch prüfen, ob diese geänderte Routeneinträge enthalten. Er vergleicht also die Einträge der Routingtabelle, welche in vorherigen Update-Zyklen bereits vom Absender der Update-Nachricht übernommen wurden, mit den aktuell übermittelten Routen des betreffenden Routers. Sind Änderungen enthalten, werden diese in die eigene Routingtabelle übernommen.

Wurde ein Eintrag in der Routingtabelle mehr als 180 Sekunden – gemessen vom Timeout-Timer – nicht aktualisiert oder bestätigt, so wird angenommen, dass das entsprechende Netzwerk nicht mehr erreichbar ist. Die Metrik zu diesem Netzwerk wird dann auf RIP-Infinity[3] gesetzt. Dies wird allen Nachbarroutern mitgeteilt. Sollte der Router für weitere 120 Sekunden – gemessen vom Garbage-Collection-Timer – keine Erreichbarkeitsinformationen zu diesem Netzwerk erhalten, wird der Eintrag aus seiner Routingtabelle gelöscht.

RIP ist ein optimaler Routingalgorithmus, wenn das Netzwerk eine Baumstruktur besitzt. Leider haben Netze mit einer solchen Topologie einen entscheidenden Nachteil: Fällt irgendwo ein Teilnetz oder ein Router aus, zerfällt das gesamte Netzwerk in mehrere Partitionen, welche gegenseitig nicht mehr erreichbar sind. Eine Lösung dieses Problems ist die Einführung von redundanten Verbindungen. Allerdings wird hierbei die Baumstruktur des Netzwerkes aufgegeben und es entstehen Schleifen. Diese führen in RIP zum sogenannten Counting to infinity Problem (CTI), welches im nächsten Kapitel beschrieben wird.

3 Infinity: unendlich. Hier gleichzusetzen mit unerreichbar.

2.1.2 Counting to infinity (CTI)

Die Entstehung eines CTIs soll zunächst an folgendem Beispiel dargestellt werden, bevor in den Kapiteln 2.1.3 und 2.1.4 Maßnahmen gegen diese Problematik angesprochen werden.

Beispiel

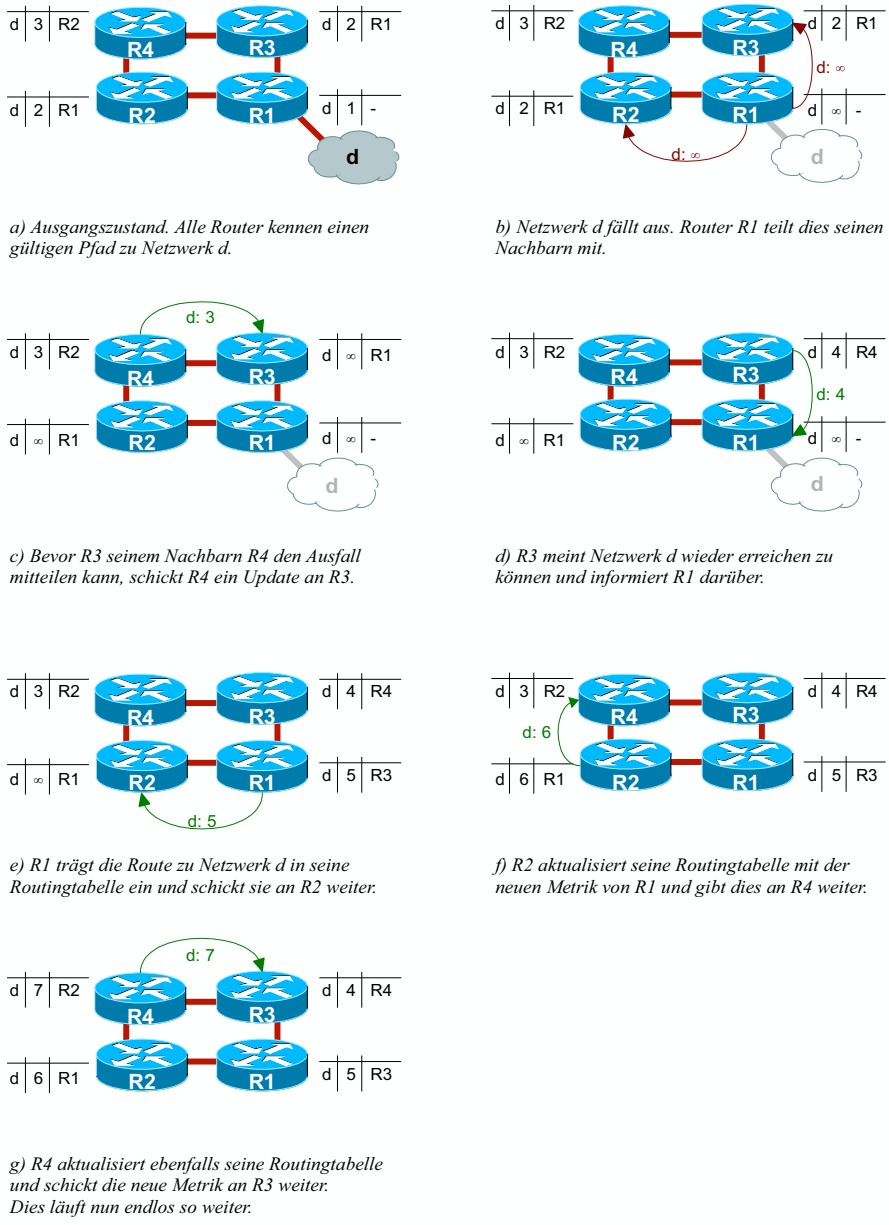


Abbildung 2.2: Entstehung eines CTIs

Angenommen es sei die in Abbildung 2.2.a gezeigte Netzwerkkonfiguration gegeben. Nun fällt Netzwerk *d* aus. Router *R1* bemerkt dies nach einiger Zeit und aktualisiert

seine Routingtabelle. Der Eintrag für Netzwerk *d* wird also auf RIP-Infinity gesetzt. Bei dem nächsten Updatezyklus teilt *R1* die Unerreichbarkeit von Netzwerk *d* den Routern *R2* und *R3* mit (Abbildung 2.2.b). Diese übernehmen ebenfalls RIP-Infinity für Netzwerk *d* in ihre Routingtabelle. Noch bevor *R2* oder *R3* ihre Informationen an *R4* weitergeben können, versendet *R4* ein Update an *R3* (Abbildung 2.2.c). Da *R4* noch nicht über die Unerreichbarkeit von Netzwerk *d* informiert wurde, enthält seine Routingtabelle immer noch einen gültigen Eintrag für das Netzwerk. Durch das Update glaubt *R3*, dass *R4* weiterhin einen Pfad zu Netzwerk *d* kennt und *R3* übernimmt diese Informationen in seine Routingtabelle. *R3* gibt nun die angenommene Erreichbarkeit von *d* an *R1* weiter (Abbildung 2.2.d), dieser wiederum an *R2* (Abbildung 2.2.e). *R2* verursacht eine Aktualisierung der Metrik von Netzwerk *d* bei *R4* (Abbildung 2.2.f). Dies gibt *R4* an *R3* weiter (Abbildung 2.2.g) und immer so fort.
Die geschaffene Routingschleife setzt sich unendlich lange fort. Jedes Mal, wenn ein Router eine neue, höhere Metrik zur Erreichbarkeit von Netzwerk *d* erhält, teilt er dies seinem nächsten in der Schleife liegenden Nachbarrouter mit.

Damit die Schleife irgendwann abbricht und das Netzwerk trotz des CTI konvergieren kann, muss der Wert RIP-Infinity, also die Metrik, ab der ein Netzwerk als unerreichbar gilt, auf einen endlichen Wert beschränkt werden. Dabei muss ein Kompromiss getroffen werden. Durch einen endlichen RIP-Infinity-Wert wird der Durchmesser des Netzwerkes beschränkt, da größere Metriken als RIP-Infinity nicht möglich sind. Wird RIP-Infinity zu klein gewählt, sind nur noch sehr kleine Netzwerke möglich. Ein großer Wert hingegen führt zu einer langen Konvergenzzeit des Netzwerkes, da der CTI wesentlich länger läuft, bis der Wert erreicht wird. Deshalb wurde RIP-Infinity auf 16 festgesetzt. Die längste gültige Route in einem Netzwerk mit RIP-Routern besitzt also eine Länge von 15.

Der CTI an sich ist durch unnötig lange Konvergenzzeiten also schon ein Problem, bringt aber auch noch ein weiteres mit sich: So glauben alle Router in der CTI-Schleife eine gültige Route zu Netzwerk *d* zu besitzen. Diese propagieren sie natürlich auch an Nachbarn außerhalb der Schleife, so dass sich diese falschen Informationen immer weiter über das Netzwerk verbreiten. Das führt dazu, dass alle Pfade, die ein Datenpaket an Netzwerk *d* durchlaufen kann, in der CTI-Schleife enden. Diese Schleife zieht also den gesamten Datenverkehr, der für das nicht mehr erreichbare Netzwerk *d* bestimmt ist, an sich und die Daten laufen – abhängig von ihrer TTL (Time To Live) – mehrfach durch die Schleife, bis sie verworfen werden. Die Netzwerkauslastung innerhalb der Schleife steigt extrem an. Hierdurch wiederum kommt es zu Kapazitätsengpässen und unter Umständen zu häufigeren Paketverlusten von RIP-Updates – insbesondere, da der Versand der Update-Nachrichten verbindungslos per UDP erfolgt, eine Übermittlungsbestä-

tigung deshalb nicht stattfindet und verlorene Pakete nicht erneut angefordert werden können. Das zieht die Dauer eines CTI unter Umständen unnötig in die Länge, deshalb wird versucht das Auftreten von CTIs zu minimieren bzw. möglichst ganz zu verhindern.

Im Folgenden werden nun einige Möglichkeiten vorgestellt, wie dies bewerkstelligt werden kann. Keine dieser Maßnahmen bietet jedoch eine komplette Verhinderung von CTIs. Deshalb ist dieses Problem auch in der zweiten RIP-Version sowie in RIPng vorhanden und bis heute ein aktuelles Forschungsthema.

2.1.3 Triggered Updates

Die Idee hinter den Triggered Updates ist neues Wissen schnellstmöglich im Netzwerk zu verbreiten. Hätte jeder Router eine globale Sicht auf das Netzwerk, das heißt alle Router wären gleichzeitig über das gesamte Netzwerk informiert, könnten keine CTIs mehr auftreten. Da es sich bei den einzelnen Routern aber um autonome Systeme handelt und nicht jeder Router jede Information direkt vorliegen hat, existiert diese globale Sichtweise nicht. Deshalb wird versucht die Zeit, in der einige Router noch falsche Informationen besitzen, möglichst zu minimieren. Dies geschieht mit Hilfe von Triggered Updates.

Bei Triggered Updates werden Änderungen in der Netzwerktopologie nicht erst nach Ablauf des Update-Intervalls weiter gemeldet, sondern sofort nach dem Eintragen der Änderungen in die eigene Routingtabelle übernommen. Dies hat den Vorteil, dass sich Änderungen an der Topologie schnell im Netzwerk verbreiten und so das Zeitfenster, in dem ein CTI entstehen kann, sehr klein wird. Allerdings wird das Netzwerk durch ständige Update-Nachrichten auch stark belastet. Deshalb wurde ein Kompromiss gefunden: Bekommt ein Router von einem seiner Nachbarn eine Update-Nachricht, welche zu Änderungen an seiner Routingtabelle führt, werden diese Änderungen direkt weiter propagiert. Gleichzeitig wird aber ein Timer gestartet und auf einen zufälligen Startwert zwischen 0 und 5 Sekunden gesetzt. In dieser Zeit eingehende Änderungen an der Routingtabelle werden erst nach Ablauf dieses Timers gebündelt weiter gesendet. So fällt die Netzwerkbelastung durch Update-Nachrichten moderater aus, allerdings erhöht sich damit auch das CTI-Risiko wieder etwas.

2.1.4 Split Horizon

Ein weiteres Feature zur Vermeidung von CTIs ist Split Horizon. Es unterbindet mögliche Routingschleifen zwischen zwei Routern. Dies geschieht mit Hilfe einer einfachen Regel: „Belehre nicht deinen Lehrer.“

Bei eingeschaltetem Split Horizon meldet ein Router die Erreichbarkeitsinformationen von Netzwerken nicht an jene Router zurück, von denen er sie übernommen und gespeichert hat.

Für den mit Split Horizon eigentlich unmöglichen Fall einer Schleife zwischen zwei Routern kann zusätzlich noch die Funktion Poisoned Reverse genutzt werden. Bei Split Horizon with Poisoned Reverse sendet der Router seinem „Lehrer“ diejenigen Routen mit der Metrik 16 zurück, die er von ihm erhalten bzw. „gelernt“ hat. Somit würde eine entstandene Routingschleife zwischen den beiden Routern direkt überschrieben werden.

Oftmals wird die Erweiterung Poisoned Reverse als unwichtig angesehen, da das Auftreten einer Schleife zwischen zwei Routern bei aktivem Split Horizon schon unmöglich erscheint. In der Softwaretechnik hingegen gehört es zum guten Programmierstil in Programmen auch die scheinbar unmöglichen Fälle zu behandeln und abzusichern ([HUN03], S. 113). Deshalb sollte Poisoned Reverse immer zusammen mit Split Horizon aktiviert werden. Beides verhindert jedoch nicht das Auftreten von CTIs zwischen mehreren Routern.

2.2 RIPv2

Heute existiert RIP in der zweiten Version, welche in RFC 2453 [RFC2453] vom November 1998 beschrieben wird und aktuell als Standard gilt.

In Version 2 des Routing Information Protocols wurden auf Basis der Version 1 fünf Neuerungen eingeführt:

Classless Interdomain Routing: RIPv2 nutzt nun das 1993 in RFC 1518 [RFC1518] und RFC 1519 [RFC1519] beschriebene Classless Interdomain Routing (CIDR). Beim CIDR wird nicht mehr die historische Aufteilung in Class-A-, Class-B- und Class-C-Netzwerke verwendet. Stattdessen wird durch die Netzmaske festgelegt, welcher Teil der Adresse das Netzwerk bezeichnet und welcher Teil die Adresse des Hosts. Hierdurch kann die maximal mögliche Hostanzahl in einem Netzwerk den tatsächlichen Gegebenheiten besser angepasst werden, so dass weniger Adressen ungenutzt bleiben.

Für RIP bedeutet die Unterstützung von CIDR, dass in den Update-Nachrichten die Netzmaske jedes Netzwerkes übermittelt werden muss. Das Nachrichtenformat der Version 2 ist in Abbildung 2.3 auf Seite 10 skizziert. Im Vergleich zum Nachrichtenformat von Version 1 (Abbildung 2.1 auf Seite 4) ist zu sehen, dass die einzelnen Routeneinträge nun nach der IP-Adresse statt Null-Bytes auch die Subnetzmaske enthalten.
Da bei RIPv2 nur die Platzhalter in den Nachrichten zusätzliche Verwendung fanden, können von RIPv2 verschickte Nachrichten auch von RIPv1 gelesen werden. Die Informationen in den nun belegten Platzhaltern werden von RIPv1 nicht beachtet.

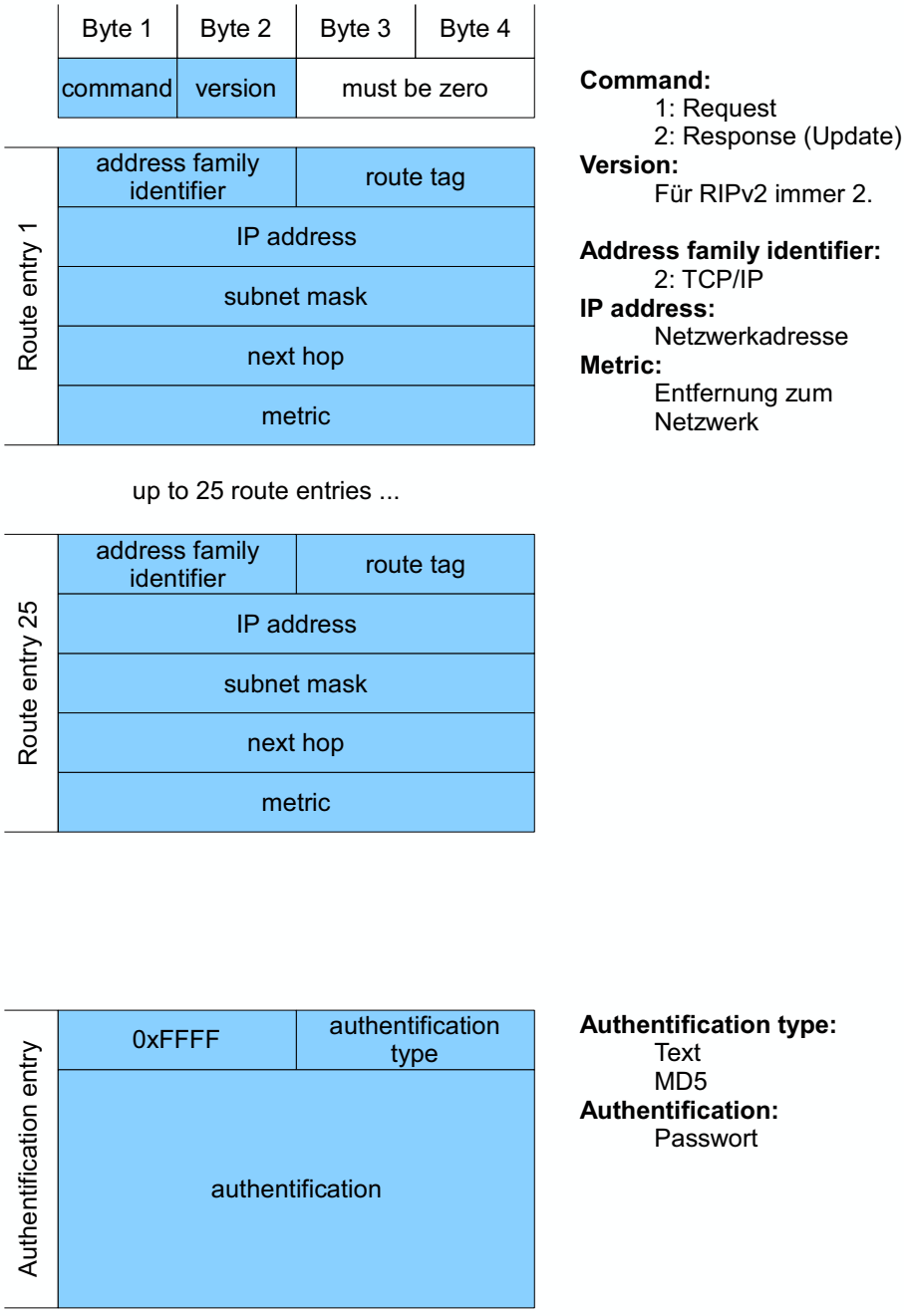


Abbildung 2.3: Nachrichtenformat RIP Version 2

Multicast: Während in Version 1 die Routingupdates per Broadcast versendet werden, nutzt RIPv2 das ressourcenschonendere Multicast [FEFE] um Update-Nachrichten auszutauschen.

Beim Multicast werden die Update-Nachrichten – genauso wie beim Broadcast – in den Netzwerkswitches vervielfältigt, so dass jede Nachricht nur einmal über jede Teilstrecke des Netzwerkes läuft. Allerdings werden die Nachrichten beim Multicast nicht an alle

Netzwerkteilnehmer weitergeleitet, sondern nur an jene, welche die Nachrichten zuvor abonniert haben. In diesem Fall werden die Nachrichten also an alle RIP-Router verteilt.

Authentifizierung: Mit der Authentifizierung soll sichergestellt werden, dass der Router Updates nur von bestimmten Nachbarn annimmt. So wird verhindert, dass die Routingtabellen sich böswillig manipulieren lassen.
Zur Übermittlung des Passwortes wird der erste Routen-Eintrag eines Nachrichtenpaketes genutzt.

RIP kennt zwei verschiedene Authentifizierungsverfahren. Bei der Klartextübermittlung des Passwortes wird dieses einfach als Klartext in die Routingupdates eingefügt und übermittelt. Es wird also keine wirkliche Sicherheit erreicht, da das Passwort jederzeit über einen „Netzwerksniffer“ wie zum Beispiel Wireshark[4] mitgelesen und anschließend missbraucht werden kann.

Als zweites Authentifizierungsverfahren kennt RIP die Authentifizierung per MD5-Hashwert (Message-Digest Algorithm 5). Hierbei wird das Passwort nicht über das Netzwerk übermittelt. Stattdessen berechnet der Router aus dem Passwort und dem Routingupdate einen MD5-Hashwert. Dieser wird in das Routingupdate eingefügt und verschickt. Der Empfänger berechnet ebenfalls aus seinem Passwort und der Nachricht einen MD5-Hashwert und vergleicht diesen mit dem in der Nachricht enthaltenen Wert. Stimmen beide überein, wird die Update-Nachricht akzeptiert.
Die Authentifizierung per MD5-Hashwert ist auf jeden Fall sicherer als die Klartextübermittlung des Passwortes und sollte deshalb immer anstelle dieser gewählt werden. Allerdings wird selbst die MD5-Hashfunktion heutzutage nicht mehr als sicher erachtet [SOT08]. Deshalb wird oft empfohlen statt MD5 den „Secure Hash Algorithm“ (SHA) zu verwenden. Dieser wird von RIP allerdings nicht unterstützt.

In Abbildung 2.3 auf Seite 10 ist unten das Nachrichtenformat von RIPv2 zur Authentifizierung notiert. Statt des Protokolltyps der normalen Routeneinträge wird der Authentifizierungseintrag durch zwei Bytes mit dem Hexadezimalwert `FFFF` gekennzeichnet. Anschließend stehen zwei Bytes für die beiden oben vorgestellten Authentifizierungstypen zur Verfügung. Die restlichen 16 Bytes dienen dem Eintrag des Passwortes bzw. des MD5-Hashwertes.

Route Tag: Mit Hilfe des Route Tags ist es möglich zusätzliche Informationen, welche nicht von RIP stammen und nicht von RIP verarbeitet werden können, über RIP Domains hinweg zu transportieren. Für das Route Tag sind in jedem Routeneintrag 2 Bytes reserviert (vgl. Abbildung 2.3).

4 Wireshark ist ein Analysetool für Netzwerkkommunikation. Die Software steht unter der GPL (GNU General Public License) und ist unter http://www.wireshark.org/ erhältlich.

Next Hop: Seit RIPv2 kann zu einem Eintrag in der Routingtabelle auch noch eine sogenannte Next Hop Adresse gespeichert werden. Das Nachrichtenformat von RIPv2 bietet in jedem Routeneintrag zwischen Subnetzmaske und Metrik eine Kapazität von 4 Bytes zur Aufnahme der Next Hop Adresse.

Die Next Hop Adresse anzugeben ist praktisch, wenn es im Netzwerk Router gibt, welche nicht per RIP kommunizieren. Diese werden von einem Router, welcher nur RIP „versteht“, nicht erkannt. Der gesamte Verkehr zu diesen Routern müsste daher über einen weiteren Router laufen, der zwischen den verschiedenen Routingprotokollen übersetzen kann.

Beispiel

Angenommen drei Router wären wie in Abbildung 2.4 durch ein Netzwerk miteinander verbunden. Auf Router *R1* läuft ausschließlich RIP, während auf Router *R3* ausschließlich OSPF (Open Shortest Path First) als Routingprotokoll zur Verfügung steht. Router *R2* beherrscht RIP sowie OSPF.

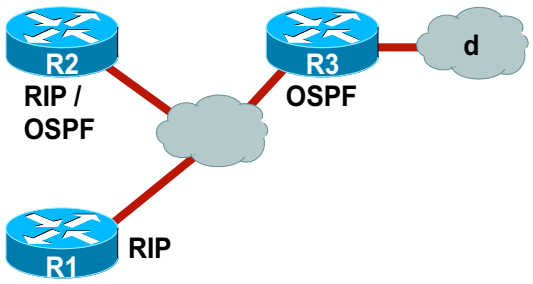


Abbildung 2.4: Mögliche Netzwerkkonfiguration zur Nutzung der Next Hop Funktionalität

R3 propagiert nun die Erreichbarkeit von Netzwerk *d*. Da *R1* OSPF nicht beherrscht, ignoriert er diese Nachricht. *R2* hingegen nimmt Netzwerk *d* in seine Routingtabelle auf und propagiert die Erreichbarkeit von Netzwerk *d* per OSPF und RIP weiter. *R1* erhält also von *R2* eine Nachricht, dass dieser Netzwerk *d* erreichen kann und übernimmt diese Information in seine eigene Routingtabelle. Ohne Next Hop Funktionalität werden somit alle eingehenden Datenpakete von *R1* an *R2* weitergeleitet. Dieser schickt sie dann an *R3* weiter.
Mit der Next Hop Funktionalität von RIPv2 kann *R2* in seinen Update-Nachrichten *R3* als Next Hop eintragen. Nun weiß *R1*, dass er Netzwerk *d* auch direkt über *R3* erreichen kann und nicht mehr den Umweg über *R2* nutzen muss.

2.3 RIPng

RIPng (ng steht für „next generation“) wird in RFC 2080 [RFC2080] vom Januar 1997 beschrieben und ist dort als „proposed standard“ markiert. Es soll hier der Vollständigkeit halber kurz erläutert werden, hat aber keine weitere Bedeutung für diese Diplomarbeit.

Bei RIPng wurde als wichtigste Neuerung – im Vergleich zu RIPv1 und RIPv2, die nur IPv4 unterstützen – der Support von IPv6 hinzugefügt. Weiterhin wird im RFC eine Änderung bezüglich des Update-Timers vorgeschlagen. Bei älteren RIP-Versionen wurden Updates in einem Intervall von 30 Sekunden ± einem Offset von 0 bis 5 Sekunden versendet. Bei RIPng wird dieser Offset auf die Hälfte der Länge des Updateintervalls vergrößert. Bei einem Updateintervall von 30 Sekunden beträgt der Offset nun ± 0 bis 15 Sekunden. Insgesamt wird also nicht mehr alle 25 bis 35 Sekunden eine Update-Nachricht versendet, sondern alle 15 bis 45 Sekunden. So soll die beim Updatevorgang entstehende Netzlast gleichmäßiger über die Zeit verteilt werden.

In Abbildung 2.5 auf Seite 14 wird das Nachrichtenformat von RIPng gezeigt. Es fällt zunächst auf, dass als Version im Header die Versionsnummer 1 angegeben wird. Dies kollidiert aber nicht mit der Versionsangabe von RIPv1. RIPng wurde entwickelt, um parallel zu RIPv1 bzw. RIPv2 eingesetzt zu werden.[5] Deshalb nutzt RIPng den Port 521, während die ursprünglichen RIP-Versionen Port 520 nutzen.

Der Next Hop ist bei RIPng nur optional, weil die Adressen bei IPv6 mit 128 Bit (16 Bytes) vier mal so lang sind wie die 32 Bit-Adressen von IPv4. Der Next Hop wird in den meisten Fällen nicht benötigt, weshalb der Eintrag standardmäßig eingespart wird. So behält die Nachricht trotz der 128 Bit langen Adresse ihre ursprüngliche Größe von 20 Bytes bei, während Routeneinträge mit Next Hop eine Größe von 40 Bytes haben. So lange kein Next Hop Eintrag in der Nachricht vorhanden ist, stellt der Versender der Nachricht den Next Hop dar.

Die Anzahl der Routeneinträge pro Update-Paket ist in RIPng nicht mehr auf maximal 25 festgesetzt. Stattdessen berechnet sich die maximale Anzahl der Routeneinträge pro Update-Paket nach Formel 2.1. Von der maximalen Paketgröße des Übertragungsmediums MTU werden die Größe des IP-Headers IP_h, die Größe des UDP-Headers UDP_h sowie die Größe des RIP-Headers RIP_h abgezogen. Diese für Daten maximal nutzbare Größe wird nun durch die Größe eines Routeneintrages RTE_s geteilt. Dieses Ergebnis wird nach der Gauß'schen Abrundungsfunktion abgerundet und stellt die maximale Anzahl an Routeneinträgen RTE_c pro Update-Paket dar. So ist sichergestellt, dass

5 Ein gleichzeitiger Betrieb von RIPng und RIPv1 bzw. RIPv2 ist notwendig, da IPv4 und IPv6 in der Umstellungsphase parallel betrieben werden und RIPng IPv4 nicht unterstützt.

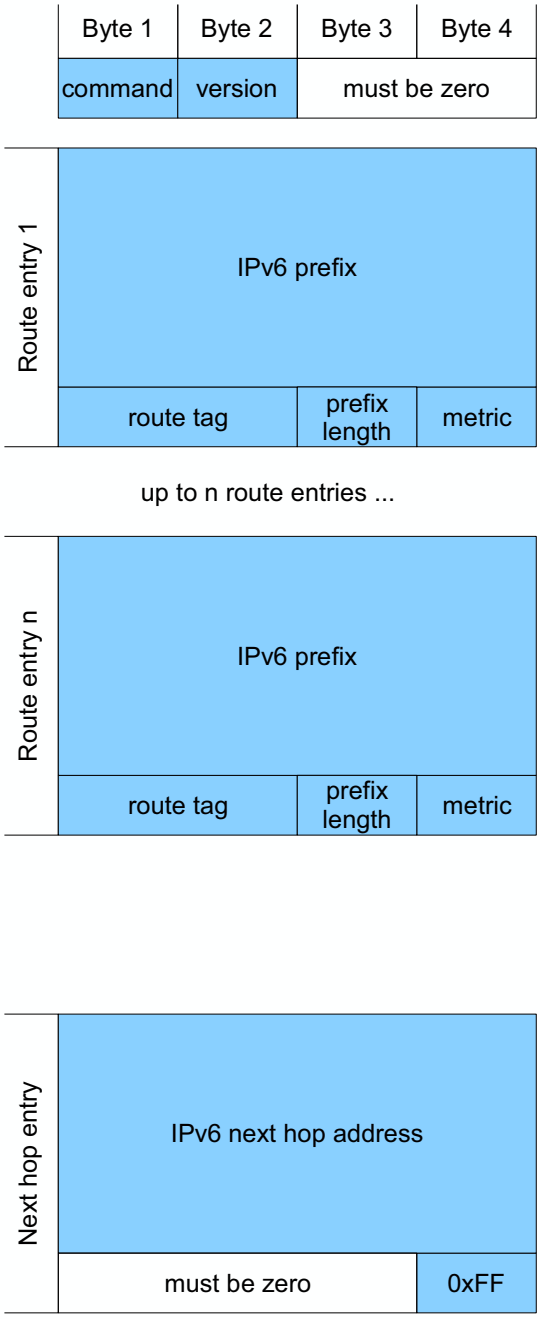


Abbildung 2.5: Nachrichtenformat RIPng

eine Update-Nachricht als Ganzes übertragen wird und nicht von einem unterhalb von UDP angesiedelten Netzwerkprotokoll aufgeteilt wird.

$$RTE_c = \left\lfloor \frac{MTU - IP_h - UDP_h - RIP_h}{RTE_s} \right\rfloor \tag{2.1}$$

Die Authentifizierung wird in RIPng über den „IP Authentication Header“ abgewickelt [RFC1826].[6]

2.4 RIP-MTI

RIP-MTI wurde 1999 unter dem Namen „Routing Information Protocol with Minimum Topology Information“ von A. Schmid [SCH99] im Rahmen seiner Diplomarbeit an der Universität Koblenz-Landau entwickelt. Heute wird die Bezeichnung „Routing Information Protocol with Metric based Topology Investigation“ bevorzugt, weil bei RIP-MTI die ohnehin schon vorliegenden Informationen der Nachbarrouter besser ausgewertet werden und durch diesen Namen die Funktionsweise des Protokolls besser beschrieben wird.

Das Ziel von RIP-MTI ist die vollständige Verhinderung von CTIs. Hierzu werden aus eingehenden Routeninformationen Rückschlüsse auf die Topologie des Netzwerkes gezogen. Dabei müssen nicht alle Router im Netzwerk RIP-MTI-Router sein, da das Nachrichtenformat von RIPv2 nicht verändert wurde und somit RIPv2 und RIP-MTI kompatibel sind. Eine Kompatibilität mit RIPv1 ist wie in Kapitel 2.2 beschrieben ebenfalls möglich. Der Betrieb von RIP-MTI zusammen mit RIPv1 ist jedoch nicht sinnvoll, da RIPv1 kein CIDR unterstützt.

Zur Auswertung der eingehenden Informationen von den Nachbarroutern gibt es bei RIP-MTI verschiedene Verfahren, die sogenannten Modes. Sie sind unterschiedlich mächtig und werden im Folgenden nun genauer vorgestellt. Die detaillierte Implementierung dieser Modes ist in [BOH08] beschrieben.

2.4.1 Normal Mode

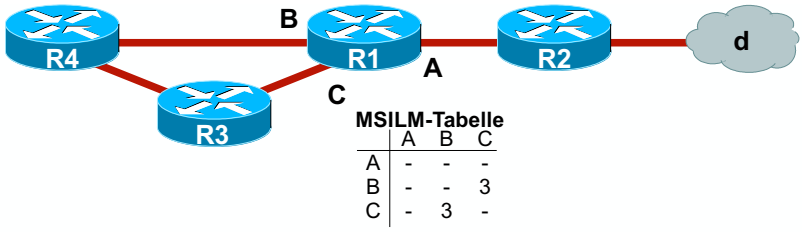
Der Normal Mode stellt den ursprünglichen Betriebsmodus von RIP-MTI dar. Er setzt eine Tabelle voraus, in der zu jedem Interface des Routers eingetragen ist, ob eine Verbindung (Schleife) zu einem weiteren eigenen Interface existiert. Zu jeder in dieser Tabelle eingetragenen Schleife wird die Metrik der kürzesten Verbindung zwischen den Interfaces – also der minimale Schleifenumfang – benötigt. In [BOH08] wird diese Tabelle als MSILM-Tabelle (Minimal Simple Loop Metrik Tabelle) bezeichnet.

Mit Hilfe der MSILM-Tabelle kann der Router nun entscheiden, ob er eine Update-Nachricht eines Nachbarrouters akzeptiert oder ignoriert. Wie dies geschieht, veranschaulicht das folgende Beispiel:

6 Weitere Informationen zum IP Authentication Header liefern u.a. RFC 2404, RFC 4302

Beispiel

Es sei die in Abbildung 2.6 gezeigte Netzwerkkonfiguration gegeben. Von besonderem Interesse ist Router *R1*. An Interface *A* liegt über *R2* Netzwerk *d* mit der Metrik 2. Dieses Interface ist mit keinem der beiden anderen Interfaces über eine Schleife verbunden. Interface *B* und *C* sind hingegen mit einer Schleife des Umfangs 3 verbunden. Diese Information hat *R1* in seiner MSILM-Tabelle gespeichert.



	A	B	C
A	-	-	-
B	-	-	3
C	-	3	-

Abbildung 2.6: Nutzung der MSILM-Tabelle zum Ablehnen von Routen.

Erhält Router *R1* nun über ein anderes Interface als Interface *A* eine Nachricht zur Erreichbarkeit von Netzwerk *d*, muss er diese ignorieren. Seine Informationen für das Treffen dieser Entscheidung bezieht er aus der MSILM-Tabelle. Dort ist von Interface *A* zu keinem anderen Interface eine Schleife eingetragen und somit kein alternativer Pfad über ein anderes Interface möglich.

Ist zwischen zwei Interfaces eine Schleife vorhanden, ist ein alternativer Pfad theoretisch möglich. Allerdings besteht bei topologischen Schleifen auch immer die Gefahr eines CTIs. Ob es sich nun wirklich um einen alternativen Pfad handelt, kann durch den Vergleich der über die beiden Interfaces gemeldeten Metriken mit dem Umfang der Schleife ermittelt werden. Dies geschieht nach Formel 2.2.
$metrik^{d}_{if1}$ und $metrik^{d}_{if2}$ bezeichnen hierbei die über die verschiedenen Interfaces gemeldeten Metriken zu Netzwerk *d*. $msilm_{if1\,if2}$ bezeichnet die Metrik der minimalen Schleife zwischen den Interfaces *A* und *B*, wie sie in der MSILM-Tabelle vermerkt ist.

$$metrik^{d}_{if1} + metrik^{d}_{if2} - 1 \geq msilm_{if1\,if2} \tag{2.2}$$

Ist die Bedingung in Formel 2.2 erfüllt, wird die Update-Nachricht als gültig angesehen und weiterverarbeitet. Anderenfalls löst sie möglicherweise einen CTI aus und wird deshalb verworfen.

Wie in [BOH08] auf Seite 76 gezeigt wird, können allerdings sehr einfach Topologien mit verschachtelten Schleifen entworfen werden, so dass die in Formel 2.2 genannte Be-

dingung erfüllt ist, aber trotzdem eine eingehende Update-Nachricht einen CTI auslösen kann.

2.4.2 Strict Mode

Um auch in komplizierten Netzwerktopologien mit verschachtelten Schleifen CTI-freie Routingupdates zu ermöglichen, wurde der Strict Mode entwickelt. Er basiert auf einem Vorschlag von T. Kleemann [KLE01] und wird in [BOH08] implementiert und erweitert.

Der Strict Mode benötigt zu jedem Interface die in [BOH08] definierte MRPM (Minimal Return Path Metrik), um korrekt zu funktionieren. Die MRPM wird aus der MSILM-Tabelle berechnet und bezeichnet für jedes Interface die Metrik der kleinsten anliegenden Schleife.

Eingehende Routingupdates werden nun nach der in Formel 2.3 genannten Bedingung überprüft (in [BOH08] Y-Test genannt). $metrik^{d}_{if1}$ bezeichnet hierbei die Metrik, mit der Netzwerk d am ursprünglichen Interface des Routers gemeldet wurde. Die über das neue Interface gemeldete Metrik von Netzwerk d wird in der Formel als $metrik^{d}_{if2}$ bezeichnet. Mit $mrpm_{if2}$ ist die MRPM des Interfaces gemeint, an dem das Netzwerk d nun gemeldet wird.

$$mrpm_{if2} + metrik^{d}_{if1} > metrik^{d}_{if2} \tag{2.3}$$

Ist die Bedingung von Formel 2.3 erfüllt, so kann durch die eingegangene Routeninformation kein CTI ausgelöst werden und die betreffende Route wird in die Routingtabelle übernommen. Ist die Bedingung nicht erfüllt, wird die eingegangene Route verworfen.

Zu beachten ist, dass sich die Metrik einer Route am aktuellen Router ändern kann, während die Routeninformationen durch eine angrenzende Schleife laufen. In diesem Fall würden für die in Formel 2.3 genannte Bedingung die falschen Voraussetzungen gelten. Deshalb muss als $metrik^{d}_{if1}$ immer die kleinste innerhalb der letzten zwei Updateintervalle gemeldete Metrik zu Netzwerk d eingesetzt werden. In [BOH08] wird dies als „verzögerter Y-Test“ bezeichnet, da hier unter Umständen mit einer veralteten Metrik getestet wird.

Mit dem oben angeführten Strict Mode konnten CTIs bei allen in [BOH08] getesteten Netzwerktopologien erfolgreich verhindert werden. Leider führt der Strict Mode bei bestimmten Netzwerktopologien jedoch zu einer langsameren Konvergenz des Netzwerkes. So werden bei gleicher Metrik auch Routingupdates abgelehnt, die nicht zum CTI

führen können und damit tatsächliche Alternativrouten zu einem ausgefallenen Netzwerk darstellen, weil sie nicht von den CTI-auslösenden Nachrichten unterschieden werden können.

2.4.3 Careful Mode

Der Careful Mode ist der Versuch das oben aufgeführte Problem zu lösen und die verzögerte Konvergenzzeit im Strict Mode zu optimieren. Entsprechend funktioniert er grundlegend genauso wie der Strict Mode. Eingehende Update-Nachrichten werden jedoch nicht mehr komplett unabhängig ausgewertet. Ist die Entscheidung, ob durch die aktuelle Update-Nachricht ein CTI ausgelöst werden kann, in diesem Moment nicht möglich, wird der Entscheidungsprozess so lange hinausgezögert, bis eine eindeutige Entscheidung getroffen werden kann.

Folgend nun die einzelnen Ansätze zur Verkürzung der Konvergenzzeit:

External Split Horizon Check (ESHC): Bei diesem Careful Mode trägt der Router eine Route aus einer eingehenden Update-Nachricht, welche möglicherweise einen CTI auslösen kann, nicht direkt in seine Routingtabelle ein. Stattdessen wartet er, bis er die Route über alle weiteren Interfaces, welche mittels einer Schleife mit dem ersten Interface verbunden sind, angeboten bekommt. Dies sollte recht schnell geschehen, wenn die Erreichbarkeitsinformationen per Triggered Update verbreitet werden.
Ist nun ein Interface des Routers über eine Schleife mit dem ersten Interface verbunden und bekommt der Router dort die Route nicht angeboten, so vermutet der ESHC-Careful-Mode eine Unterdrückung der Update-Nachricht durch aktives Split Horizon bei seinem entsprechenden Nachbarrouter. Dies würde bedeuten, dass der Nachbarrouter die betreffende Route vom aktuellen Router übernommen hat. In diesem Fall hätte aber der Nachbarrouter seine Erreichbarkeitsinformationen in die Schleife geschickt, so dass ein CTI auftritt. Deshalb dürfen zu dieser Route eingehende Update-Nachrichten nicht weiterverarbeitet werden.

Nach [BOH08] konvergiert das Netzwerk im ESHC-Careful-Mode zwar schneller als im Strict Mode, allerdings ist der Speicher- und Rechenaufwand zur Prüfung einer Route recht hoch. So muss für jede Route eine Liste geführt werden, über welche Interfaces sie bereits gemeldet wurde. Hierdurch werden beträchtliche Ressourcen verbraucht, vor allem wenn sich an vielen Routen Änderungen ergeben.

Deny Timer (DT): Der DT-Careful-Mode ist grob gesagt die Umkehrung des ESHC-Careful-Modes. Er wartet nicht, bis eine Route über alle weiteren durch eine Schleife verbundenen Interfaces angeboten wird. Stattdessen wird bei einer möglichen CTI-auslösenden Update-Nachricht auf allen durch eine Schleife mit diesem Interface verbundenen Interfaces ein RIP-Infinity für die entsprechende Route verschickt. Die eingehenden

Routen-Informationen werden für einen gewissen Zeitraum zwischengespeichert. Sollten sie nach Ablauf des vom Deny Timer definierten Zeitraumes nicht durch RIP-Infinity überschrieben worden sein, werden sie in die Routingtabelle übernommen.

Im DT-Careful-Mode wird also blind angenommen, dass mindestens ein Nachbarrouter seine Informationen über die gerade behandelte Route vom aktuellen Router bezieht. Es wird allerdings nicht versucht den oder die entsprechenden Nachbarrouter zu identifizieren. Durch das Senden des RIP-Infinitys wird beim Nachbarrouter eine eventuell auf den aktuellen Router zeigende Route zu dem entsprechenden Netzwerk einfach überschrieben. Dies sollte sich – sofern eine Route zum aktuellen Router vorhanden war – durch die Schleife fortsetzen, so dass die ursprünglichen Routeninformationen nach kurzer Zeit durch RIP-Infinity überschrieben werden.

Die Zwischenspeicherung der vom Deny Timer behandelten Routen verbraucht – wenn viele Routen behandelt werden – auch in diesem Mode recht viele zusätzliche Ressourcen. Das muss als Nachteil gewertet werden, auch wenn im Vergleich zum ESHC-Careful-Mode weniger Ressourcen benötigt werden.

Request Timer (RT): Der RT-Careful-Mode arbeitet prinzipiell genauso wie der DT-Careful-Mode. Bei einer eingehenden, möglicherweise einen CTI-auslösenden Route wird in die angrenzenden Netzwerkschleifen ein RIP-Infinity gesendet. Im Unterschied zum DT-Careful-Mode wird allerdings die zu prüfende Route nicht lokal zwischengespeichert, sondern verworfen. Nach Ablauf des Request Timers wird die Route per Request-Nachricht vom Nachbarrouter neu angefordert.

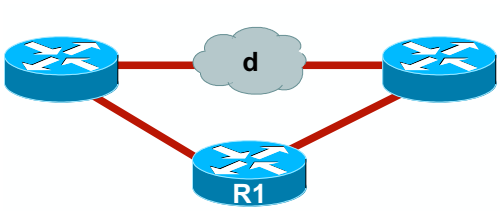
Da im RT-Careful-Mode die Routen nicht lokal gespeichert werden müssen, geht dieser Mode am schonendsten mit den verfügbaren Ressourcen um und wird deshalb in [BOH08] als bevorzugter Betriebsmodus empfohlen.

2.4.4 Schleifenerkennung

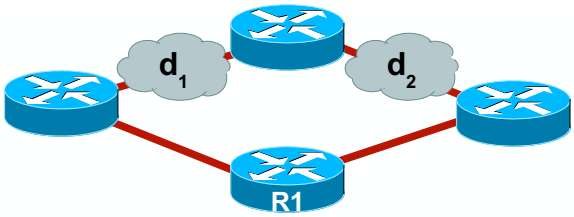
Wie im Kapitel 2.4.1 dargestellt, benötigt der Normal Mode von RIP-MTI die MSILM-Tabelle, um seine Entscheidungen zu treffen. Der RIP-MTI-Strict-Mode sowie die Careful Modes benötigen die MRPM, welche sich aus der MSILM-Tabelle berechnen lässt (siehe Kapitel 2.4.2).

Ein zentrales Problem von RIP-MTI ist also die Erstellung der MSILM-Tabelle und damit die Erkennung der dort einzutragenden topologischen Schleifen.

Abbildung 2.7 auf Seite 20 zeigt die beiden möglichen topologischen Schleifentypen. Der linke Teil der Abbildung zeigt eine Schleife mit einer ungeraden Anzahl an Routern bzw. Netzwerken, der rechte eine Schleife mit einer geraden Anzahl an darin liegenden Netzwerken.



a) Ungerader Schleifenumfang. Die Schleife kann an einem Netzwerk erkannt werden.



b) Gerader Schleifenumfang. Es existieren zwei Netzwerke, durch welche eine Erkennung der Schleife möglich ist.

Abbildung 2.7: Erkennung von topologischen Schleifen

In der linken Schleife bekommt Router *R1* Netzwerk *d* über seine beiden Nachbarrouter jeweils mit der Metrik 2 gemeldet. Die Schleife selbst hat den Umfang 3.

Bei der rechten Schleife in Abbildung 2.7 bekommt Router *R1* die Netzwerke d_1 und d_2 jeweils über zwei verschiedene Nachbarrouter gemeldet. Netzwerk d_1 wird über eines der Interfaces von *R1* mit Metrik 2, über das andere Interface mit der Metrik 3 gemeldet. Analog gilt dies für Netzwerk d_2. Die Schleife selbst hat einen Umfang der Metrik 4.

Je nach Topologie existieren also ein oder zwei Netzwerke, welche dem Router *R1* über zwei verschiedene Interfaces gemeldet werden. Alle anderen Netzwerke in der Schleife sind jeweils nur über eines der beiden Interfaces „sichtbar“, weil RIP nur den Pfad mit der kürzesten Metrik in seiner Routingtabelle speichert und weiter propagiert.

Da es bei einer Schleife mit geradem Umfang egal ist, an welchem der beiden Netzwerke (d_1 oder d_2) sie nun erkannt wurde, soll zukünftig auch bei solchen Schleifen nur noch allgemein gesprochen werden, dass die Schleife durch Netzwerk *d* erkannt wurde.

Es ist nun leicht einzusehen, dass der Umfang einer Schleife immer genau eins kleiner ist als die Summe der über die beiden Interfaces gemeldeten Metriken von Netzwerk *d*. Mathematisch notiert ergibt sich so Formel 2.4. $metrik^{d}_{if1}$ und $metrik^{d}_{if2}$ bezeichnen die über die beiden Interfaces gemeldeten Metriken von Netzwerk *d*, $schleifenumfang^{d}_{if1\,if2}$ den dazu gehörenden Schleifenumfang.

Um zu prüfen, ob es sich bei dem über zwei Interfaces gemeldeten Netzwerk wirklich um das dem Router gegenüberliegende Netzwerk *d* handelt, wird bei RIP-MTI Formel 2.5 verwendet. $metrik^{d}_{if1}$ und $metrik^{d}_{if2}$ bezeichnen auch in dieser Formel die Metriken von Netzwerk *d*, wie sie über die beiden Interfaces gemeldet werden. Diese Metriken müssen entweder gleichgroß sein (ungerader Schleifenumfang) oder sie dürfen sich maximal um eins unterscheiden (gerader Schleifenumfang).

$$schleifenumfang^{d}_{if1\,if2} = metrik^{d}_{if1} + metrik^{d}_{if2} - 1 \tag{2.4}$$

$$2 > |metrik^{d}_{if1} - metrik^{d}_{if2}| \tag{2.5}$$

Da die Erkennung der Schleife nur über ein bzw. zwei Netzwerke möglich ist, ist es für den RIP-MTI-Algorithmus von zentraler Bedeutung diese Netzwerke über verschiedene Interfaces zu erkennen. Sollten zum Beispiel Routeninformationen zur Erreichbarkeit von Netzwerk *d* verloren gehen oder sogar absichtlich gefiltert werden[7], kann der Algorithmus die Schleife nicht erkennen. Es ist zwar möglich, dass eine größere Schleife erkannt wird, welche die Schleife, in der Netzwerk *d* liegt, einschließt. Genauso wäre es hingegen auch möglich die Schleife an einem günstig „hinter“ der Schleife gelegenen Netzwerk zu erkennen (siehe Abbildung 2.8). In beiden Fällen wäre aber der Schleifenumfang nicht – wie von RIP-MTI gefordert – der minimale Umfang der Schleife.

Wird der RIP-MTI-Algorithmus unter solchen Voraussetzungen betrieben, können weiterhin CTIs auftreten. Um dies zu vermeiden wird in Kapitel 3 ein Ansatz zur verbesserten Schleifenerkennung entwickelt.

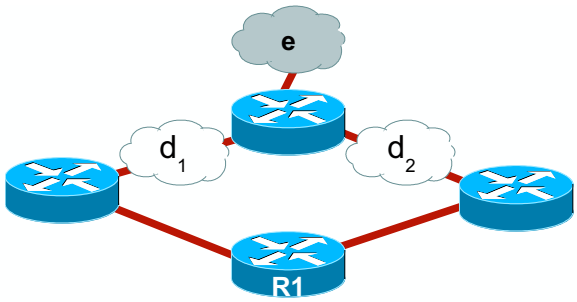


Abbildung 2.8: Schleifenerkennung durch ein hinter der Schleife liegendes Netzwerk. Die Netzwerke d_1 und d_2 sind für Router R1 nicht sichtbar. Er bekommt aber Netzwerk e über zwei Interfaces jeweils mit der Metrik 3 gemeldet. Nach Formel 2.4 nimmt R1 den Schleifenumfang nun mit der Metrik 5 statt 4 an.

7 Erreichbarkeitsinformationen über Netzwerke, die nur zur Durchleitung von Daten gedacht sind, werden gerne zur Sicherheit von den Borderroutern herausgefiltert. Wenn das Netzwerk nach außen nicht bekannt ist und somit kein gültiger Pfad zu dem Netzwerk existiert, kann die Infrastruktur nicht von außen angegriffen werden.

3 Performanz von RIP-MTI

Nachdem nun die Grundlagen von RIP und RIP-MTI bekannt sind, wird die Performanz von letzterem genauer betrachtet. Hierbei sind sowohl die Zeitdauer der Schleifenerkennung als auch die Auswirkungen von Problemen bei der Schleifenerkennung von besonderem Interesse. Weiterhin wird ein Ansatz vorgestellt, mit dem die initiale Schleifenerkennung verbessert wird.

3.1 Zeitdauer der Schleifenerkennung

Funktionsweise und Bedeutung der Schleifenerkennung für RIP-MTI sind in Kapitel 2.4.4 ausführlich beschrieben worden. Um die Schleifenerkennung von RIP-MTI besser zu verstehen, ist nun von Interesse, wie lange es dauert, bis der Router eine neu entstandene Schleife zwischen seinen Interfaces erkennt. Damit der Algorithmus zukünftig effizient eingesetzt werden kann interessiert weiterhin, wann der Algorithmus nach dem Start des Routers einsatzbereit ist, wann also alle relevanten Schleifen erkannt wurden.

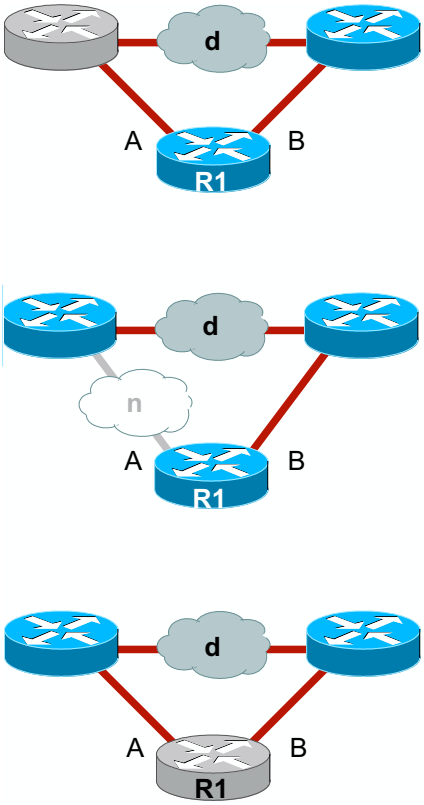
3.1.1 Entstehung topologischer Schleifen

Bevor die Zeitdauer zur Erkennung einer Schleife genauer betrachtet werden kann, muss erst einmal geklärt werden, wie eine topologische Schleife im Netzwerk überhaupt entstehen kann.

Sei Router *R1* wie in Abbildung 3.1 auf Seite 23 der RIP-MTI-Router, welcher die topologische Schleife erkennen soll. Zur Entstehung topologischer Schleifen gibt es aus Sicht von *R1* drei verschiedene Szenarien, die in der genannten Abbildung skizziert sind:

1. Szenario: Ein neuer Router wird in das Netzwerk integriert. Dieser Router verbindet mehrere Netzwerke, so dass nun ein Pfad von Interface *A* zu Interface *B* des Routers *R1* entsteht, also eine Schleife vorliegt.
2. Szenario: Zwei Router werden mit einem neuen Netzwerk verbunden. Hierdurch entsteht eine Verbindung zwischen zwei Interfaces von Router *R1*. Als Sonderfall dieses Szenarios kann das Schließen der Schleife durch Netzwerk *d*, welches Router *R1* zur Schleifenerkennung nutzt, angesehen werden.
3. Szenario: Der Router *R1* wird so in das Netzwerk integriert, dass eine Schleife im Netzwerk entsteht.

Auf den ersten Blick scheint es sich bei dem dritten Szenario um einen Sonderfall von Szenario 1 zu handeln. Bei der später erfolgenden genaueren Betrachtung der einzelnen



Szenario 1:
Ein neuer Router wird in das Netzwerk integriert und schließt eine Schleife.

Szenario 2:
Zwei Router werden durch ein neues Netzwerk miteinander verbunden.

Szenario 3:
Router R1 wird in das Netzwerk integriert und schließt mindestens eine Schleife.

Abbildung 3.1: Skizzen der Szenarien bei der Entstehung topologischer Schleifen. Hell dargestellt jeweils der neue Router bzw. das neue Netzwerk.

Szenarien zeigt sich aber, dass sich gerade dieses dritte Szenario von den beiden anderen deutlich abhebt, weshalb es hier gesondert aufgeführt wird.

3.1.2 Schleifenerkennung in den Szenarien 1 und 2

Szenario 1 und 2 sind vom Ablauf der Schleifenerkennung her fast identisch. Von dem Spezialfall in Szenario 2 abgesehen, ist in beiden Fällen Router *R1* das Netzwerk *d* bereits über ein Interface bekannt. Nun muss die Erreichbarkeitsinformation von Netzwerk *d* nur noch von der Stelle, an der die Schleife geschlossen wurde, bis zum Router *R1* transportiert werden, damit die Schleife erkannt wird. Die für diese Strecke benötigte Zeit entspricht der Zeitdauer für die Schleifenerkennung.

Handelt es sich um eine Schleife mit einer geraden Anzahl an Netzwerken, kann die Schleife durch zwei verschiedene Netzwerke erkannt werden. Es wird angenommen, dass die Erkennung der Schleife durch das Netzwerk geschieht, welches Router *R1* bereits über ein Interface bekannt ist, wenn die Schleife durch eines der beiden zur Erkennung nutzbaren Netzwerke geschlossen wird.

Im Sonderfall, dass es nur ein Netzwerk *d* gibt und die Schleife durch genau dieses Netzwerk geschlossen wurde, kann es *R1* noch nicht bekannt sein. Daher müssen die

Erreichbarkeitsinformationen zu Netzwerk *d* erst über zwei Interfaces an Router *R1* gemeldet werden, bis die Schleife erkannt werden kann. Dabei weisen beide Verbindungen die gleiche Streckenlänge auf. Es ist deshalb zu erwarten, dass die Erreichbarkeitsinformationen zu Netzwerk *d* ungefähr gleichzeitig eintreffen.

3.1.3 Vorüberlegungen zur Berechnung der Zeitdauer bis zur Schleifenerkennung in Szenario 1 und 2

Eine allgemeine Formel, wie sich die Zeitspanne t_{gesamt} vom Schließen der Schleife bis zu deren Erkennung zusammensetzt, kann sehr leicht angegeben werden (Formel 3.1). Die Latenzzeit ist die Zeit, die benötigt wird, um die Informationen über das Netzwerk zu übertragen. Weiterhin ergibt sich an jedem auf der Strecke liegenden Router *R* eine Verzögerung t_R.

$$t_{gesamt} = Latenz + \sum_{R=1}^{n} t_R, \quad n \in \mathbb{N} \wedge n < \text{RIP-Infinity} \tag{3.1}$$

Diese Verzögerung kann verschiedene Ursachen haben:

Verzögerung bei Triggered Updates: Wie in Kapitel 2.1.3 beschrieben, wird bei Triggered Updates die erste eingehende Update-Nachricht direkt weiter verschickt. In diesem Fall ist t_R null. Mit dem Versand einer getriggerten Update-Nachricht wird der Timer auf einen zufälligen Wert zwischen 0 und 5 Sekunden gesetzt. Das nächste Triggered Update kann erst nach Ablauf des Timers verschickt werden. t_R nimmt also bei aktivierten Triggered Updates einen Wert zwischen 0 und 5 Sekunden an. Jedoch sind nicht alle Werte gleichwahrscheinlich. Die Verzögerung kann nicht größer als die Laufzeit des zufällig gewählten Timers sein. Trifft die nächste Update-Nachricht jedoch nicht direkt am Startzeitpunkt des Timers ein, ist die Verzögerung kürzer als die initiale Laufzeit des Timers. Hieraus folgt, dass kürzere Verzögerungen wahrscheinlicher sind als längere Verzögerungen. Die Wahrscheinlichkeitsdichtefunktion für die Verzögerung durch den Timer sieht also wie in Abbildung 3.2 dargestellt aus.

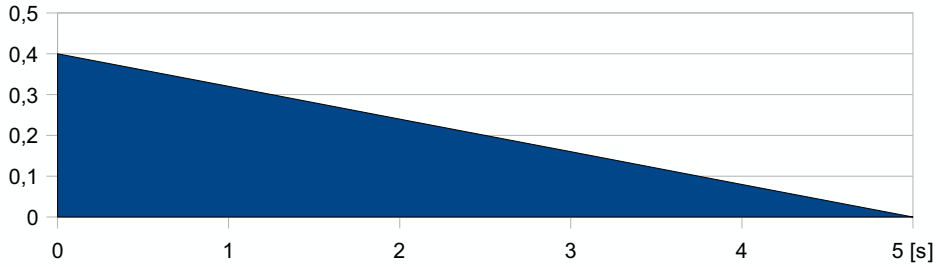


Abbildung 3.2: Wahrscheinlichkeitsdichtefunktion der Verzögerung durch den Timer bei Triggered Updates.

Zusätzlich muss nun zur Berechnung der gesamten zu erwartenden Verzögerung noch beachtet werden, ob überhaupt eine Verzögerung durch den Timer stattfindet oder ob die Routinginformationen direkt weitergeleitet werden. Deshalb spielt bei der Berechnung des Erwartungswertes für die Verzögerung auch die Wahrscheinlichkeit eine Rolle, dass der Timer überhaupt gestartet wird. Die Wahrscheinlich hierfür wird folgend mit p_t bezeichnet.

Insgesamt ergibt sich als Erwartungswert für die Verzögerung durch Triggered Updates Formel 3.2. Die 5/3 (1,67) Sekunden sind der Erwartungswert gemäß Abbildung 3.2 für die Verzögerung durch den Timer. Eine ausführliche Herleitung des Erwartungswertes sowie der Berechnung der Standardabweichung ist in Anhang A zu finden.

$$\mu = p_t \cdot \frac{5}{3}\text{sec.}, \quad 0 \le p_t \le 1 \tag{3.2}$$

Da p_t vom jeweiligen Netzwerk abhängt, kann hierfür keine allgemeingültige Standardabweichung angegeben werden. Entsprechend lässt sich diese für Formel 3.2 auch nicht berechnen.

Verzögerung bei Timed Updates: Ein Timed Update erfolgt alle 25 bis 35 Sekunden (vgl. Kapitel 2.1). Der Versand der Timed Updates ist also – im Gegensatz zu den Triggered Updates – vollkommen unabhängig von eingehenden Update-Nachrichten. Die Update-Nachricht kann im besten Fall direkt weiter versendet werden. Ist jedoch gerade ein Timed Update erfolgt, müssen die Informationen der eingegangenen Update-Nachricht schlimmstenfalls 35 Sekunden auf ihre Weiterverbreitung warten. t_R nimmt hier entsprechend einen zufälligen Wert zwischen 0 und 35 Sekunden an, wobei jedoch die Wahrscheinlichkeiten nicht gleichmäßig verteilt sind. Es ergibt sich die in Abbildung 3.3 dargestellte Wahrscheinlichkeitsverteilung. Verzögerungen von 0 bis 25 Sekunden sind unabhängig vom Offset des Update-Timers immer möglich. Größere Verzögerungen können sich nur ergeben, wenn der Update-Timer auch mit einem mindestens solchen Wert initialisiert wurde. Entsprechend nimmt die Wahrscheinlichkeit bei Verzögerungen von mehr als 25 Sekunden linear ab.

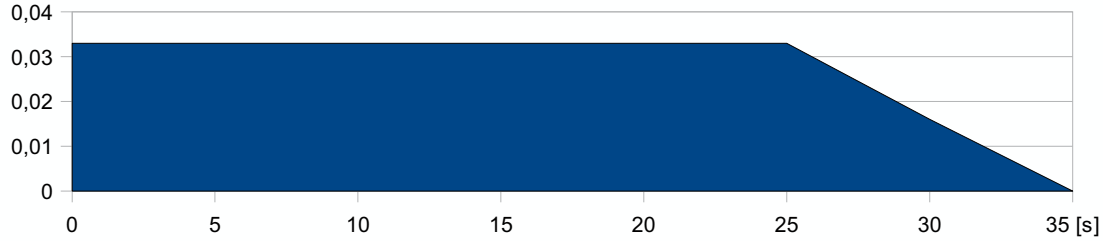


Abbildung 3.3: Wahrscheinlichkeitsverteilung der Verzögerung bei Timed Updates.

Durch diese Wahrscheinlichkeitsverteilung ergibt sich für die Verzögerung bei Timed Updates ein Erwartungswert von 545/36 (15,14) Sekunden (Berechnung sowie Standardabweichung siehe Fehler: Referenz nicht gefunden).

Verzögerung durch verlorene Update-Nachrichten: Geht eine Update-Nachricht verloren – egal, ob es sich um ein Triggered oder ein Timed Update handelt – sind die zu übermittelnden Informationen im nächsten Timed Update erneut enthalten.
Für jedes beim Nachbarrouter nicht eingetroffene Timed Update verlängert sich die Übermittlung der Informationen also um die Länge des nächsten Update-Intervalls (25 bis 35 Sekunden). Ginge ein Triggered Update verloren, kann die Verzögerung auch kürzer ausfallen, da Triggered Updates außer der Reihe ausgeführt werden. Diese Kombination von Verzögerungen wird jedoch erst später behandelt.

Die maximale Verzögerung durch verlorene Update-Nachrichten liegt auf jeden Fall bei 180 Sekunden. Sollte nämlich, wie in Kapitel 2.1 beschrieben, ein Router über einen Zeitraum von mehr als 180 Sekunden keine Nachrichten von seinem Nachbarrouter empfangen, so werden alle bislang über diesen Router erreichbaren Netzwerke als unerreichbar markiert. Diese Unerreichbarkeit wird umgehend an alle Nachbarrouter weiter propagiert. Somit ist die Schleife zumindest in einer Richtung für die Router aufgehoben. Ein anschließender erfolgreicher Austausch von Update-Nachrichten kann dann als erneutes Schließen der topologischen Schleife angesehen werden. Je nachdem, ob ein Router oder ein Netzwerk für den Verlust der Update-Nachrichten verantwortlich war, handelt sich sich um Szenario 1 oder 2.

Der Erwartungswert für die Verzögerung ist in diesem Fall von der Wahrscheinlichkeit für den Verlust einer Update-Nachricht abhängig. Je höher die Wahrscheinlichkeit liegt, desto größer ist die zu erwartende Verzögerung. So lange keine Nachrichten im Netzwerk verloren gehen, entstehen hierdurch auch keine Verzögerungen. Geht durchschnittlich jede zweite Nachricht verloren ($p_v = 0{,}5$), ist mit einer Verzögerung von etwa 30 Sekunden zu rechnen. Kommt nur jede dritte Nachricht an ($p_v = 0{,}66$), beträgt die Verzögerung schon 60 Sekunden. Im konkreten Fall können innerhalb der 180 Sekunden maximal 6 Update-Nachrichten verloren gehen (bei einem zu erwartenden Update-Intervall von 30 Sekunden[8]). Spätestens die siebte Update-Nachricht muss also ihr Ziel erreichen, damit der Timeout-Timer des Nachbarrouters nicht abläuft. Dies würde einem durchschnittlichen Paketverlust von 86% entsprechen, also einer Wahrscheinlichkeit p_v von 0,86 für den Verlust einer Update-Nachricht. Tabelle 3.1 zeigt die entsprechenden Daten.

8 Es handelt sich um eine stetige Gleichverteilung. Die Berechnung des Erwartungswertes und der Standardabweichung sind in Anhang C aufgeführt.

p_v	0	0,5	0,66	0,75	0,8	0,83	0,86
Verzögerung [s]	0	30	60	90	120	150	180

Tabelle 3.1: Erwartete Verzögerungen bei verschiedenen Wahrscheinlichkeiten für den Verlust von Update-Nachrichten.

Bei genauer Betrachtung der Werte fällt auf, dass die Verzögerung für $p_v \to 1$ gegen unendlich geht. Gehen alle Update-Nachrichten verloren, ist also $p_v = 1$, findet eine Übermittlung der Informationen niemals statt.
Da die Kehrwertfunktion $f(x) = 1/x$ für $x \to 0$ ebenfalls gegen unendlich geht, liegt es nahe von dieser Funktion auszugehen, um die zu Tabelle 3.1 gehörende Funktion zu ermitteln. In Anhang D ist ausführlich dargestellt, wie sich Formel 3.3 aus der Kehrwertfunktion durch Streckung, Verschiebung und Spiegelung herleiten lässt.

Somit kann nach Formel 3.3 die zu erwartende Verzögerung bei verlustbehafteter Nachrichtenübertragung berechnet werden.

$$\mu = \left(\frac{1}{1 - p_v} - 1 \right) \cdot 30\,\text{sec.}, \quad 0 \le p_v < 1 \tag{3.3}$$

3.1.4 Berechnung der Zeitdauer bis zur Schleifenerkennung für Szenario 1 und 2

Mit den Vorüberlegungen aus Kapitel 3.1.3 kann nun recht schnell ein minimaler und ein maximaler Wert sowie der Erwartungswert für die Zeitdauer vom Schließen der topologischen Schleife bis zur Erkennung derselben durch RIP-MTI angegeben werden.

Der Übersicht halber wird jedoch zuerst noch ein Router besonders gekennzeichnet: Der Router R_d. Er liegt zwischen Router *R1* und Netzwerk *d*, und zwar dort, wo die topologische Schleife geschlossen wird. Er befindet sich am nächsten an *R1* und kennt dabei noch Netzwerk *d*, wobei sein Pfad zu diesem Netzwerk jedoch nicht über Router *R1* verläuft (siehe Abbildung 3.4 links). Die Erreichbarkeitsinformationen zu Netzwerk *d* müssen also die Strecke von R_d bis *R1* durchlaufen, damit *R1* die Schleife erkennen kann.
Für den Sonderfall, dass die topologische Schleife durch Netzwerk *d* geschlossen wird (siehe Abbildung 3.4 rechts auf Seite 28) und keinem Router das Netzwerk *d* bereits bekannt ist, sollen die beiden Borderrouter von Netzwerk *d* als R_d bezeichnet werden. Die beiden Strecken zwischen R_d und *R1* sind hier jeweils gleich lang, so dass keine Fallunterscheidung stattfinden muss.

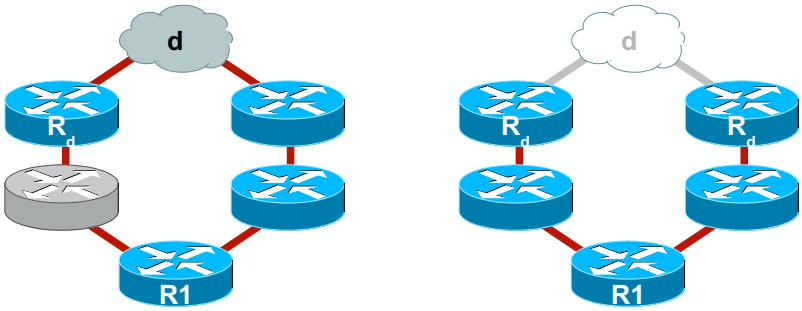


Abbildung 3.4: Definition von Router R_d.
Router R_d stellt denjenigen Router dar, welchem Netzwerk d noch bekannt ist, bevor die topologische Schleife geschlossen wird. Im Sonderfall, dass die Schleife durch Netzwerk d geschlossen wird, werden die beiden Borderrouter von Netzwerk d als R_d definiert.

Die **minimale Laufzeit** ergibt sich, wenn jeder zwischen R_d und *R1* liegende Router die Informationen über Netzwerk *d* sofort weiter propagiert. Die Laufzeit der Informationen würde in diesem Fall allein von der Laufzeit auf der Leitung abhängen, also der Latenzzeit entsprechen (siehe Formel 3.4).

$$t_{min} = Latenz \tag{3.4}$$

Bei der Berechnung der **maximalen Laufzeit** sowie des **Erwartungswertes** müssen verschiedene Fälle unterschieden werden:

1. **Kein Verlust von Update-Nachrichten:** Hierbei ergibt sich die maximale Laufzeit der Informationen, wenn jeder Router unmittelbar vor Eintreffen der Update-Nachricht bereits eine eigene Update-Nachricht versendet hat. In diesem Fall muss das komplette Intervall des Timers gewartet werden, bis die nächste Update-Nachricht verschickt werden kann. Hinzu kommt natürlich noch die Latenzzeit. Für die Nutzung von Triggered und Timed Updates ergeben sich verschiedene Werte:

   1.1. **Nutzung von Triggered Updates:** Wie im vorherigen Kapitel aufgeführt, können sich bei der Nutzung von Triggered Updates pro Router Verzögerungen von maximal 5 Sekunden ergeben. Daraus folgt für die maximale Laufzeit Formel 3.5. c_R stellt die Anzahl der Router zwischen R_d und *R1* dar und nimmt entsprechend Werte zwischen 0 und RIP-Infinity an.

   Der Erwartungswert für die Verzögerung an einem Router wurde bereits in Kapitel 3.1.3 (Formel 3.2) berechnet. Nun muss dieser nur noch mit c_R, der Anzahl der Router, multipliziert und die Latenzzeit hinzuaddiert werden. Als Erwartungswert für die Dauer der Schleifenerkennung ergibt sich somit der in Formel 3.6 berechnete Wert.

$$t_{max} = Latenz + c_R \cdot 5\,\text{sec.}, \quad c_R \in \mathbb{N} \wedge c_R < \text{RIP-Infinity} \tag{3.5}$$

$$t_{\mu} = Latenz + c_R \cdot p_t \cdot \frac{5}{3}\,\text{sec.}, \quad \begin{cases} c_R \in \mathbb{N} \wedge \\ c_R < \text{RIP-Infinity} \wedge \\ 0 \le p_t \le 1 \end{cases} \tag{3.6}$$

1.2. **Nachrichtenübermittlung per Timed Updates:** Die maximale Verzögerung beträgt 35 Sekunden je Router, so dass sich analog zu der Nachrichtenübermittlung per Triggered Updates Formel 3.7 ergibt.
Mit den Vorüberlegungen aus dem vorherigen Kapitel ist die Berechnung des Erwartungswertes in diesem Fall genauso einfach wie im vorherigen. Als Erwartungswert für die Verzögerung an einem Router wurden in Kapitel 3.1.3 545/36 Sekunden berechnet. Entsprechend ergibt sich für die Zeit, die die Informationen von R_d nach *R1* benötigen, der Erwartungswert aus Formel 3.8.

$$t_{max} = Latenz + c_R \cdot 35\,\text{sec.}, \quad c_R \in \mathbb{N} \wedge c_R < \text{RIP-Infinity} \tag{3.7}$$

$$t_{\mu} = Latenz + c_R \cdot \frac{545}{36}\,\text{sec.}, \quad c_R \in \mathbb{N} \wedge c_R < \text{RIP-Infinity} \tag{3.8}$$

2. **Verlust von Update-Nachrichten:** In Kapitel 3.1.3 wurde bei den Möglichkeiten zur Verzögerung der Updates beschrieben, dass die Weiterleitung der Informationen um maximal 180 Sekunden verzögert wird, wenn Update-Nachrichten verloren gehen. Somit ergibt sich Formel 3.9 als maximale Zeitdauer zur Übertragung der Informationen von R_d nach *R1*. Dieser Wert hat jedoch eher theoretische Bedeutung. Es ist in der Praxis extrem unwahrscheinlich, dass über einige Router hinweg jeweils mehrere Update-Nachrichten verloren gehen, bis die Informationen weiter übertragen werden.

   Der Erwartungswert für die Übertragung der Erreichbarkeitsinformationen ergibt sich, indem die zu erwartende Verzögerung beim Verlust eines bestimmten Anteils der Update-Nachrichten (Formel 3.3) mit c_R multipliziert und die Latenzzeit hinzu addiert wird (Formel 3.10).

$$t_{max} = Latenz + c_R \cdot 180\,\text{sec.}, \quad c_R \in \mathbb{N} \wedge c_R < \text{RIP-Infinity} \tag{3.9}$$

$$t_{\mu} = Latenz + c_R \cdot \left[\left(\frac{1}{1 - p_v} - 1 \right) \cdot 30\,\text{sec.} \right], \quad \begin{cases} c_R \in \mathbb{N} \wedge \\ c_R < \text{RIP-Infinity} \wedge \\ 0 \le p_v < 1 \end{cases} \tag{3.10}$$

Hierbei muss allerdings beachtet werden, dass in diesem Abschnitt *ausschließlich* die Verzögerung durch den Verlust von Update-Nachrichten behandelt wird. Eine Verzögerung durch Triggered bzw. Timed Updates wird hierbei *nicht* betrachtet. Dies erfolgt erst später. Hier wird also davon ausgegangen, dass die Router synchron laufen und Update-Nachrichten immer nur zu den Zeitpunkten eingehen, an denen ein Timed Update ausgeführt wird.

Nachdem die Verzögerungen in den einzelnen Fällen getrennt berechnet wurden, sind nun in Hinsicht auf die Berechnung der Zeitdauer zur Schleifenerkennung zwei Kombinationen der verschiedenen Verzögerungsarten von 1. und 2. interessant: Erstens der Erwartungswert für die Dauer der Informationsübertragung, wenn die Verbreitung der Routinginformationen per Timed Update geschieht und hierbei ein gewisser Anteil der Update-Nachrichten verloren geht, und zweitens der gleiche Erwartungswert bei der Nutzung von Triggered Updates. Hierbei können ebenfalls Update-Nachrichten verloren gehen. Eine erneute Übermittlung der Informationen geschieht dann mit dem nächsten Timed Update.

Für den ersten Fall ist die gesamte zu erwartende Zeitdauer recht einfach zu berechnen. Zum einen werden die Informationen durch die Timed Updates verzögert, so dass sich der Erwartungswert aus Formel 3.8 ergibt Durch die Verluste der Update-Nachrichten verzögert sich die Übermittlung der Informationen jedoch weiter. Diese zusätzliche zu erwartende Verzögerung kann nach Formel 3.10 ermittelt werden. Der gesamte Erwartungswert ergibt sich aus der Summe der beiden Teilergebnisse, natürlich ohne Verdopplung der Latenzzeit (siehe Formel 3.11).

$$t_\mu = Latenz + c_R \cdot \left(\frac{545}{36}\,\text{sec.} + \left[\left(\frac{1}{1-p_v} - 1 \right) \cdot 30\,\text{sec.} \right] \right), \quad \begin{cases} c_R \in \mathbb{N} \wedge \\ c_R < \text{RIP-Infinity} \wedge \\ 0 \le p_v < 1 \end{cases} \tag{3.11}$$

Im zweiten Fall, wenn Triggered Updates genutzt werden, ist die Sache etwas komplizierter. Auf jeden Fall werden zur Berechnung des gesamten Erwartungswertes die Berechnungen aus Formel 3.6 benötigt. Wird zu diesem Wert jedoch wie im vorangegangen Fall einfach der Erwartungswert aus Formel 3.10 hinzuaddiert, ergibt sich – ebenfalls ohne die doppelte Latenzzeit – ein zu hoher Erwartungswert. Dies liegt daran, dass bei dem Verlust eines Triggered Updates die erneute Versendung der Informationen nicht erst nach 25 bis 35 Sekunden erfolgt. Da Triggered Updates außer der Reihe versendet werden, kann die Zeitdauer bis zum nächsten Timed Update zwischen 0 und 35 Sekunden betragen. In Abbildung 3.3 wurde hierfür bereits die Wahrscheinlichkeitsverteilung vorgestellt, so dass sich für das Intervall zwischen einem Triggered Update und dem nachfolgenden Timed Update eine zu erwartende Zeitspanne von 545/36

Sekunden ergibt. Entsprechend können von dem Wert aus Formel 3.10 einfach die 535/36 Sekunden ($30\,\text{Sekunden} - 545/36\,\text{Sekunden}$), die zu viel berechnet wurden, abgezogen werden. Leider funktioniert dies so nur bei höheren Verlustwahrscheinlichkeiten von $p_v \geq 0{,}5$. Ist die Wahrscheinlichkeit für einen Paketverlust jedoch sehr gering ($p_v < 1/3$), würde diese Formel einen negativen Erwartungswert für die Verzögerung liefern. Dies kann jedoch nicht sein. Deshalb muss eine Fallunterscheidung vorgenommen werden. Bei niedriger Wahrscheinlichkeit für den Verlust von Update-Nachrichten wird in Formel 3.10 nun direkt mit 545/36 Sekunden statt mit 30 Sekunden multipliziert. Dies ist möglich, da bei wenigen Paketverlusten meist gar nicht so viele Nachrichten verloren gehen, dass mehrere Updateintervalle hintereinander die Update-Nachrichten ihr Ziel nicht erreichen.

Durch die abschnittsweise Definition ergibt sich jedoch wieder ein neues Problem. Zuvor wurde mit der durchschnittlichen Wahrscheinlichkeit für Paketverluste im gesamten Netzwerk gerechnet. Wenn diese sich aber nun in den einzelnen Teilnetzen stark unterscheidet – was anzunehmen ist – entstehen ungenaue Erwartungswerte. Deshalb müssen die Erwartungswerte für alle Teilnetze getrennt berechnet und aufsummiert werden, so dass sich nun für den Erwartungswert bei Nutzung von Triggered Updates und Verlusten von Update-Nachrichten Formel 3.12 ergibt.

$$t_{v_R} = \begin{cases} \left(\frac{1}{1-p_{v_R}} - 1\right) \cdot \frac{545}{36}\,\text{sec.} & \text{für } 0 \leq p_{v_R} < 0{,}5 \\ \left(\frac{1}{1-p_{v_R}} - 1\right) \cdot 30\,\text{sec.} - \frac{535}{36}\,\text{sec.} & \text{für } 0{,}5 \leq p_{v_R} < 1 \end{cases}$$

$$t_\mu = Latenz + \sum_{R=1}^{c_R} p_t \cdot \frac{5}{3}\,\text{sec.} + t_{v_R}, \quad \begin{cases} c_R \in \mathbb{N} \wedge \\ c_R < \text{RIP-Infinity} \wedge \\ 0 \leq p_t \leq 1 \end{cases} \tag{3.12}$$

Nicht eintreten kann der Fall, dass die Informationen eines verloren gegangenen Triggered Updates durch ein nachfolgend ausgelöstes weiteres Triggered Update übertragen werden. Das liegt daran, dass der Router nach dem Senden eines solchen Updates die betreffenden Routen nicht mehr als geändert in seiner Routingtabelle führt und diese somit in späteren Triggered Updates nicht mehr überträgt.

Nun sind alle benötigten Teilformeln vorhanden und eine Gesamtformel zur Berechnung der Zeitdauer vom Schließen einer Schleife bis zu deren Erkennung durch RIP-MTI kann angegeben werden.
An dieser Stelle muss jedoch zwischen den Szenarien 1 und 2 unterschieden werden. Wird die Schleife wie in Szenario 1 durch einen neuen Router geschlossen, sendet die-

ser einen initialen Request an Router R_d. Durch die direkte Antwort von R_d werden die Erreichbarkeitsinformationen von Netzwerk *d* direkt per Triggered Update bis zu *R1* weitergeleitet. Unter Umständen können dabei Paketverluste auftreten, die zu weiteren Verzögerungen führen. Die gesamte Zeitdauer lässt sich durch Formel 3.12 berechnen.

Wurde die Schleife jedoch wie in Szenario 2 durch ein neues Netzwerk geschlossen, gibt es kein Request, um die Informationen direkt bei R_d abzufragen. Entsprechend wird die Erreichbarkeit von Netzwerk *d* erst mit dem nächsten Timed Update an den neuen Nachbarrouter weiter geschickt. Danach erfolgt der weitere Versand der Informationen per Triggered Update bis zu *R1*. Gehen Update-Nachrichten verloren, muss unterschieden werden, ob das Timed Update von R_d verloren geht, oder ob eines der Triggered Updates der anderen Router verschwindet. Insgesamt ergibt sich also eine Kombination aus den Formeln 3.11 und 3.12 (Formel 3.13).

Verzögerung bei Verlust des ersten Timed Updates:

$$t_{v_1} = \left(\frac{1}{1 - p_{v_1}} - 1 \right) \cdot 30\,\text{sec.}, \quad \text{für } 0 \le p_{v_1} < 1$$

Verzögerung bei Verlust der Triggered Updates:

$$t_{v_R} = \begin{cases} \left(\frac{1}{1 - p_{v_R}} - 1 \right) \cdot \frac{545}{36}\,\text{sec.} & \text{für } 0 \le p_{v_R} < 0{,}5 \\ \left(\frac{1}{1 - p_{v_R}} - 1 \right) \cdot 30\,\text{Sec.} - \frac{535}{36}\,\text{sec.} & \text{für } 0{,}5 \le p_{v_R} < 1 \end{cases} \tag{3.13}$$

Berechnung des Erwartungswertes:

$$t_\mu = \mathit{Latenz} + \frac{545}{36}\,\text{sec.} + t_{v_1} + \sum_{R=2}^{c_R} p_t \cdot \frac{5}{3}\,\text{sec.} + t_{v_R}, \quad \begin{cases} c_R \in \mathbb{N} \wedge \\ 2 \le c_R < \text{RIP-Infinity} \wedge \\ 0 \le p_t \le 1 \end{cases}$$

Zum Abschluss des Kapitels nun noch ein kleines Beispiel, welches die Anwendung der zuvor aufgestellten Formeln zeigt:

Beispiel

Es sei die in Abbildung 3.5 gezeigte Topologie gegeben. Router R_d ist Netzwerk *d* bereits bekannt. Nun soll berechnet werden, wie lange es minimal bzw. maximal dauert, bis Router *R1* über die Erreichbarkeit von Netzwerk *d* informiert wird. Weiterhin soll der Erwartungswert berechnet werden. Die Latenzzeit von 12 ms (0,012 Sekunden) sei bekannt. Im mittleren der drei zwischen R_d und *R1* liegenden Teilnetze liegt eine

Abbildung 3.5: Beispieltopologie: Berechnung der Zeitspanne zur Informationsübertragung von R_d nach R1.

recht hohe Paket-Verlustrate von 70 % vor. In den restlichen beiden Teilnetzen gehen lediglich 10 % der übermittelten Pakete verloren.
Bei der Übertragung der Informationen per Triggered Update sollen 30 % der Updates durch den Timer verzögert werden.

Die Berechnung des Erwartungswertes erfolgt für den Fall, dass die Strecke zwischen R_d und *R1* durch einen neuen Router geschlossen wurde. Anschließend wird die zu erwartende Zeitdauer berechnet, wenn die Verbindung der Router durch ein neues Netzwerk zustande kam. Nach Definition von R_d muss dieses neue Netzwerk bzw. der neue Router in Abbildung 3.5 direkt rechts von R_d liegen!

Zuerst wird nach Formel 3.4 von Seite 28 die minimale Laufzeit der Informationen berechnet. Durch die angenommene Latenzzeit ergibt sich hierfür eine Zeitdauer von 0,012 Sekunden.

Die theoretische Maximalzeit für die Übertragung wird nach Formel 3.9 von Seite 29 berechnet. Da Verzögerungen an Router R_d sowie an den beiden unbenannten Routern entstehen können, muss für c_R der Wert 3 eingesetzt werden. Somit ergibt sich eine maximale Zeitdauer von 540,012 Sekunden (ca. 9 Minuten).

Nun wird die zu erwartende Verzögerung berechnet, wenn die Strecke zwischen R_d und *R1* durch einen neuen Router geschlossen wurde. Dies geschieht nach Formel 3.12 von Seite 31. Hierbei werden zuerst die Teilergebnisse t_{v_R} für die vorgegebenen Verlustwahrscheinlichkeiten von 0,1 sowie 0,7 berechnet. Für erstere ergibt sich ein Wert von 1,68 Sekunden, für letztere ein Wert von 55,14 Sekunden. Da die Wahrscheinlichkeit p_t für die Verzögerung von Triggered Updates mit 0,3 vorgegeben ist, kann nun der Erwartungswert berechnet werden. Es ergibt sich eine zu erwartende Zeitdauer von 60,015 Sekunden für die Übertragung der Erreichbarkeitsinformationen von Netzwerk *d*.

Zuletzt erfolgt nach Formel 3.13 die Berechnung des Erwartungswertes für die Verzögerung, wenn die Verbindung durch ein neues Netzwerk erfolgte. Für das Netzwerk rechts von R_d, in dem die Übertragung per Timed Update erfolgt, verzögert die Verlustwahrscheinlichkeit von 0,1 die Übertragung der Informationen um 3,33 Sekunden. In den beiden anderen Netzwerken mit den Verlustwahrscheinlichkeiten von 0,7 und 0,1 ergeben sich die schon zuvor berechneten Verzögerungen von 55,14 Sekun-

den sowie 1,68 Sekunden. Insgesamt ist in diesem Fall mit einer Zeitdauer von 76,3 Sekunden für die Informationsübertragung von R_d nach *R1* zu rechnen.

3.1.5 Erkennung der topologischen Schleifen in Szenario 3

Nun erfolgt die Betrachtung des dritten Szenarios. Es wird deutlich, weshalb dieser Fall in Kapitel 3.1.1 nicht als Sonderfall von Szenario 1 angenommen wurde.

Die Nachbarrouter von *R1* besitzen hier bereits Erreichbarkeitsinformationen zu Netzwerk *d*. Wird *R1* in das Netzwerk integriert und die Schleife geschlossen, sendet er seinen Nachbarn automatisch ein Request und fordert ihre Routingtabelle an. Sobald beide in der Schleife liegenden Nachbarrouter geantwortet haben, „kennt“ *R1* Netzwerk *d* über zwei Interfaces und kann die Schleife erkennen. In diesem Szenario können also Schleifen erkannt werden, sobald *R1* die Informationen aller seiner Nachbarrouter erhalten hat. So lange keine Request- oder Update-Nachrichten verloren gehen, hängt die Zeitdauer bis zur Erkennung der Schleifen lediglich von der Antwortzeit des langsamsten Nachbarrouters ab.

Im RFC werden bezüglich des Antwortverhaltens von Request-Nachrichten keine zeitlichen Vorgaben gemacht. In der Regel sollten jedoch Request-Nachrichten von einem Router schnellstmöglich (direkt) beantwortet werden.

Für den Fall, dass entweder das Request oder das Response verloren geht, kann die zu erwartende Antwortzeit nach Formel 3.12 berechnet werden, wobei für c_R der Wert 1 einzusetzen ist. p_t ist null, da Requests immer direkt beantwortet werden und kein Timer gestartet wird. Es ergibt sich Formel 3.14.

$$t_\mu = Latenz + \begin{cases} \left(\frac{1}{1-p_{v_r}} - 1\right) \cdot \frac{545}{36}\,\text{sec.} & \text{für } 0 \le p_{v_r} < 0{,}5 \\ \left(\frac{1}{1-p_{v_r}} - 1\right) \cdot 30\,\text{sec.} - \frac{535}{36}\,\text{sec.} & \text{für } 0{,}5 \le p_{v_r} < 1 \end{cases} \tag{3.14}$$

3.2 Probleme bei der Schleifenerkennung

In den vorherigen Kapiteln wurde gezeigt, dass die Erkennung einer Schleife einige Zeit in Anspruch nehmen kann. Ab und zu kommt es sogar vor, dass RIP-MTI eine Schleife gar nicht erkennt. In diesem Kapitel werden die Ursachen für dieses Problem erörtert und die daraus resultierenden Auswirkungen auf RIP-MTI abgeschätzt. Zuletzt wird versucht eine Lösung für das Problem zu finden.

3.2.1 Ursachen

In einer Schleife gibt es je nach Umfang ein bzw. zwei Netzwerke (nachfolgend allgemein mit *d* bezeichnet), mittels derer der Router die Schleife erkennen kann. Gelingt es RIP-MTI jedoch nicht eine Schleife zu erkennen, kann dies – wie in Kapitel 2.4.4 bereits angedeutet – verschiedene Ursachen haben:

Verlust von Nachrichten: Der Nachrichtenversand bei RIP wird per UDP (User Datagram Protocol), also verbindungslos, abgewickelt. Daher ist nicht sichergestellt, dass eine verschickte Nachricht ihr Ziel erreicht. Gehen auf einem der beiden möglichen Pfade zwischen dem RIP-MTI-Router und Netzwerk *d* die zugehörigen Erreichbarkeitsinformationen verloren, kann die Schleife nicht erkannt werden.

Filterung von Nachrichten: Es kann sein, dass Erreichbarkeitsinformationen zu einem oder mehreren Netzwerken absichtlich nicht weiterpropagiert werden. Diese Filterung erfolgt aus Sicherheitsaspekten oder weil Netzwerk *d* ein nicht öffentlicher Adressraum zugewiesen wurde.

3.2.2 Auswirkungen auf RIP-MTI

Im Allgemeinen können bei RIP-MTI – genauso wie auch bei RIP – CTIs auftreten, wenn Netzwerkschleifen nicht bzw. nicht mit dem minimalen Umfang erkannt werden. Bei der Betrachtung der Auswirkungen geht es also in erster Linie darum, das Risiko für das Auftreten eines CTIs in Abhängigkeit von der jeweiligen Ursache abzuschätzen.

Verlust von Nachrichten: In Kapitel 2.1 wurde gezeigt, dass in jeder Update-Nachricht maximal 25 Routen übermittelt werden. Gehen einzelne Update-Nachrichten verloren, werden bei einem Updatevorgang mit vielen Routen wahrscheinlich nur manche Routeneinträge nicht übermittelt. Bei einem späteren Updatevorgang gehen vermutlich nicht exakt die gleichen Update-Nachrichten verloren, daher ist es sehr unwahrscheinlich, dass ein RIP-MTI-Router eine Schleife auf Grund von Verlusten bei den Update-Nachrichten nicht erkennt. Die Erkennung der Schleife kann allerdings – abhängig von der Verlustrate der Update-Nachrichten – verzögert stattfinden (siehe Kapitel 3.1.4). Da die restlichen Routen aber von der gleichen Verlustrate betroffen sind, werden sie ähnlich verzögert weiter propagiert, so dass es kaum zu einem CTI kommen kann.

Sind auf Grund eines geraden Schleifenumfangs zwei Netzwerke zur Erkennung der Schleife vorhanden, kann es sein, dass die Erreichbarkeitsinformationen zu diesen beiden Netzwerken in getrennten Update-Nachrichten weiterpropagiert werden. In diesem Fall sinkt die Wahrscheinlichkeit noch weiter, dass die Schleife erst verzögert oder gar nicht erkannt wird.

Insgesamt ist dieses Problem also nicht so schlimm, wie ursprünglich vermutet wurde. Als Auswirkung ergibt sich zwar eine verzögerte Einsatzbereitschaft von RIP-MTI. Das Risiko für das Auftreten eines CTIs ist hierdurch jedoch auch sehr gering einzuschätzen.

Filterung von Nachrichten: Da hier von einer dauerhaften Filterung auszugehen ist, ist es dem RIP-MTI-Algorithmus in keiner Weise möglich die Schleife (mit dem minimalen Umfang) zu erkennen (siehe auch Kapitel 2.4.4). In diesem Fall liegt das gleiche Risiko für das Auftreten von CTIs vor wie bei RIP.

3.2.3 Lösungsansätze für die Probleme der Schleifenerkennung

Verlust von Nachrichten: Bei Schleifen mit einem ungeraden Umfang ist an der geringen Wahrscheinlichkeit für das Auftreten eines CTIs nichts zu ändern. Bei Schleifen mit geradem Schleifenumfang hingegen könnte die Wahrscheinlichkeit für eine verzögerte oder fehlgeschlagene Erkennung noch weiter gesenkt werden. Da zur Erkennung von Schleifen mit geradem Umfang zwei Netzwerke zur Verfügung stehen, könnte durch geschickte Aufteilung der Erreichbarkeitsinformationen auf die einzelnen Update-Nachrichten erreicht werden, dass diese beiden Netzwerke *immer* in verschiedenen Update-Nachrichten weiterpropagiert werden. So müssten die Update-Nachrichten für beide Netzwerke verloren gehen, bis die Erkennung der Schleife sich verzögert oder fehlschlägt.

Eine Trennung der Erreichbarkeitsinformationen zu den beiden Netzwerken könnte gewährleistet werden, indem jeder Router innerhalb einer Update-Nachricht nur Netzwerke mit der gleichen Metrik weiterpropagiert. Da die beiden zur Schleifenerkennung geeigneten Netzwerke auf verschiedenen Seiten eines Routers liegen, müssen sie sich in ihrer Metrik jeweils um 1 unterscheiden.

Nachteilig ist jedoch der durch das Verfahren erstehende zusätzliche Aufwand. So ist die Routingtabelle eines Routers normalerweise so optimiert, dass er im Routingbetrieb den Namen eines Netzwerkes leicht in der Tabelle findet. Um einen nach Metriken getrennten Versand von Update-Nachrichten zu realisieren, müsste die Routingtabelle entweder umsortiert werden oder es müssten entsprechende Indizes eingeführt werden. Beides verursacht einen erheblichen Aufwand.
Weiterhin könnte es zu Problemen mit „normalen“ RIP-Routern kommen, da diese keine Sortierung der Netzwerke vornehmen und hierdurch die Vorteile des Verfahrens wieder zunichte gemacht werden würden.

Insgesamt lohnt sich der Mehraufwand nicht, um eine solch geringfügig kleinere Wahrscheinlichkeit für das Auftreten eines CTIs zu erreichen. Deshalb wird dieser Lösungsansatz nicht weiterverfolgt.

Filterung von Nachrichten: Eine Lösung dieses Problems scheint auf den ersten Blick nicht möglich. Im nachfolgenden Kapitel wird jedoch ein Ansatz geliefert, wie eine Erkennung der Schleife unter bestimmten Voraussetzungen dennoch erfolgen kann.

3.3 Slow Start

Die Slow Start Erweiterung wurde mit dem Ziel entwickelt topologische Netzwerkschleifen mit einem RIP-MTI-Router sicher zu erkennen und CTIs noch öfter zu vermeiden.

3.3.1 Vorüberlegungen

Zuerst einige allgemeine Vorüberlegungen, die für das Verständnis der Funktionsweise von Slow Start notwendig sind.

Die Erkennung topologischer Schleifen könnte bedeutend verbessert werden, wenn eine Schleife nicht nur an einem oder zwei Netzwerken erkannt würde. Ideal wäre eine Erkennung mittels jedes in der Schleife liegenden Netzwerkes. Da RIP und damit auch RIP-MTI jedoch immer nur die jeweils beste Route zu einem Netzwerk speichert und weiterverbreitet, ist dies nicht möglich. Allerdings wurde in Kapitel 3.1.5 gezeigt, dass die Nachbarrouter eines noch inaktiven Routers *R1* jeweils alle Netzwerke der zukünftigen Schleife kennen.

Zur Erinnerung noch einmal der Ablauf eines Routerstarts: Wird der Router eingeschaltet, verschickt er Request-Nachrichten auf allen Ports, um möglichst schnell an die Routingtabellen seiner Nachbarn zu gelangen. Sobald er ein angeschlossenes Netzwerk erkennt oder eine Antwort von einem Nachbarn erhält, beginnt er diese gewonnenen Informationen per Triggered Update im Netzwerk weiter zu verbreiten. Ist durch den neuen Router eine topologische Schleife entstanden, übernehmen seine Nachbarrouter zwangsläufig einige dieser Informationen und überschreiben hierdurch das zuvor vorhandene Wissen.
Wäre *R1* aber in der Lage nach seinem Start die kompletten Routingtabellen seiner Nachbarrouter abzufragen ohne deren Informationen vorher teilweise überschrieben zu haben, könnte er alle von RIP-MTI für die Verhinderung von CTIs benötigten Schleifen durch jedes darin liegende Netzwerk erkennen.

Weiterhin wird es in der Praxis eher selten vorkommen, dass ein komplettes Netzwerk in einem Schritt aktiviert wird. Üblicher ist es das Netzwerk nach und nach unter kontrollierten Bedingungen einzurichten und erst wenn der aktuell aktive Teil problemlos läuft, werden weitere Teile hinzu geschaltet. Genauso verhält es sich, wenn ein Netzwerk evolutionär wächst. Neue Netze und Subnetze werden im laufenden Betrieb hinzugefügt. Dabei soll das bestehende Netzwerk möglichst nicht in seiner Funktionsweise

beeinträchtigt werden. Bei der aktuellen Implementation von RIP bzw. RIP-MTI wird dies so noch nicht berücksichtigt.
Deshalb kann beispielsweise folgendes passieren: Der Router hat beim Start die Routingtabelle eines Nachbarrouters erhalten und weiterverbreitet. Nur wenige Augenblicke nach der Verbreitung seiner aktuellen (noch unvollständigen) Routingtabelle erhält er von einem anderen Router eine Antwort auf sein Request, welche neue und teilweise auch bessere Routen enthält. Diese werden dann per Triggered Update schnellstmöglich an das gesamte Netzwerk weiter verbreitet. So verursacht der Start eines Routers eine regelrechte Flut an Update-Nachrichten.

Tabelle 3.2 verdeutlicht noch einmal die zuvor gemachten Überlegungen und stellt die Unterschiede zwischen der Startphase und der anschließenden Betriebsphase eines Routers heraus. Ungeachtet dieser gravierenden Unterschiede wird bei aktuellen Routern keine Unterscheidung zwischen diesen beiden Phasen vorgenommen.

Startphase	Betriebsphase
■ Routinginformationen sind nur teilweise vorhanden. ■ Nur direkt angeschlossene Netzwerke sind zu Beginn der Startphase bekannt. ■ Routinginformationen von den Nachbarroutern werden gesammelt, wodurch sich häufige Änderungen in der Routingtabelle ergeben. ■ Nachbarrouter werden umgehend per Triggered Update über Änderungen in der Routingtabelle informiert.	■ Routingtabelle (wahrscheinlich) vollständig. ■ Austausch von Routinginformationen mit den Nachbarroutern (regelmäßig per Timed Update sowie bei Änderungen der Routingtabelle als Triggered Update).

Tabelle 3.2: Vergleich der Start- und Betriebsphase eines Routers.

3.3.2 Funktionsweise

Die Funktionsweise von Slow Start ergibt sich aus den Vorüberlegungen und ist extrem einfach. In der Startphase des Routers werden alle seine Update-Nachrichten an die Nachbarrouter unterdrückt. Es wird lediglich das initiale Request-Kommando verschickt. Nachdem alle Nachbarrouter den Request beantwortet haben, besitzt der neue Router eine vollständige Routingtabelle. Nun kann er seinen normalen Betrieb aufnehmen und seine Routingtabelle an die Nachbarrouter propagieren.

Ursprünglich war geplant das Antwortverhalten des Routers für jedes Interface einzeln zu behandeln. So können zum Beispiel Update-Nachrichten auf Interfaces, an denen kein Router zuhört, eingespart werden. Die Erkennung, ob hinter einem Interface noch ein Router vorhanden ist, erscheint auf den ersten Blick recht einfach. So könnte der Router auf allen Interfaces, auf denen er keine Nachrichten von einem Nachbarrouter

erhalten hat, statt der kompletten Routingtabelle lediglich Request-Nachrichten verschicken. Sobald das Request beantwortet wird, verschickt der Router zukünftig auf dem entsprechenden Interface seine Routingtabelle. Bekommt er für einen gewissen Zeitraum keine Nachrichten über ein bestimmtes Interface, verschickt er statt der Routingtabelle nur noch Request-Nachrichten. Hierbei muss jedoch sichergestellt sein, dass sich die Router nicht gegenseitig ignorieren und keine Update-Nachrichten mehr schicken. Dies könnte zu Unerreichbarkeiten von Teilnetzen führen und wäre fatal. Die hierfür notwendige Forschungsarbeit konnte im Rahmen dieser Arbeit nicht geleistet werden. Deshalb beschränkt sich diese Diplomarbeit auf die einfachere Variante, bei der alle Interfaces gleich behandelt werden. Hierdurch ist der Kern des Verfahrens bereits abgedeckt und die Optimierung kann bei Bedarf durchgeführt werden.

In Abbildung 3.6 auf Seite 40 ist die detaillierte Funktionsweise von Slow Start grafisch dargestellt. Nach dem Einschalten und Initialisieren verschickt der Router auf allen Interfaces Request-Nachrichten. Anschließend wartet er auf eingehende Nachrichten. Empfängt er eine Request-Nachricht, so antwortet er mit seiner Routingtabelle, auch wenn er sich in der Startphase befindet und die Routingtabelle noch unvollständig ist. Dies ist notwendig, da das Request-Kommando zum einen für Diagnose- und Monitoringzwecke gedacht ist, zum anderen wird das Request-Kommando von neuen Routern verwendet, um die Routingtabellen der Nachbarrouter abzufragen. Der entsprechende Nachbarrouter, welcher die Request-Nachricht verschickt hat, hat also ebenfalls eine leere bzw. unvollständige Routingtabelle. Ist die Routingtabelle des Nachbarrouters leer, kann dort nichts überschrieben werden. Befinden sich noch weitere Router an dem entsprechenden Interface, stellt dies kein Problem dar, weil die Antworten auf Request-Nachrichten nicht per Multicast, sondern per Unicast beantwortet werden. Sie erreichen also nur das Gerät, welches nach der Routingtabelle gefragt hat.

Empfängt der Router eine Update-Nachricht, trägt er die hierin enthaltenen Informationen in seine Routingtabelle ein. Hat sich die Routingtabelle geändert, prüft der Router, wie er nun weiter verfahren muss. Ist Slow Start aktiviert und befindet es sich in der Startphase, werden keine Update-Nachrichten verschickt. In allen anderen Fällen werden die Änderungen an der Routingtabelle per Triggered Updates weiterpropagiert.

Ähnlich verhält sich der Router, wenn der Update-Timer abgelaufen ist und ein Timed Update verschickt werden soll. Hier wird bei aktiviertem Slow Start geprüft, ob sich der Router noch in der Startphase befindet. Für diesen Fall wird das Timed Update unterdrückt. In allen anderen Fällen findet ein normaler Versand der Update-Nachrichten statt.

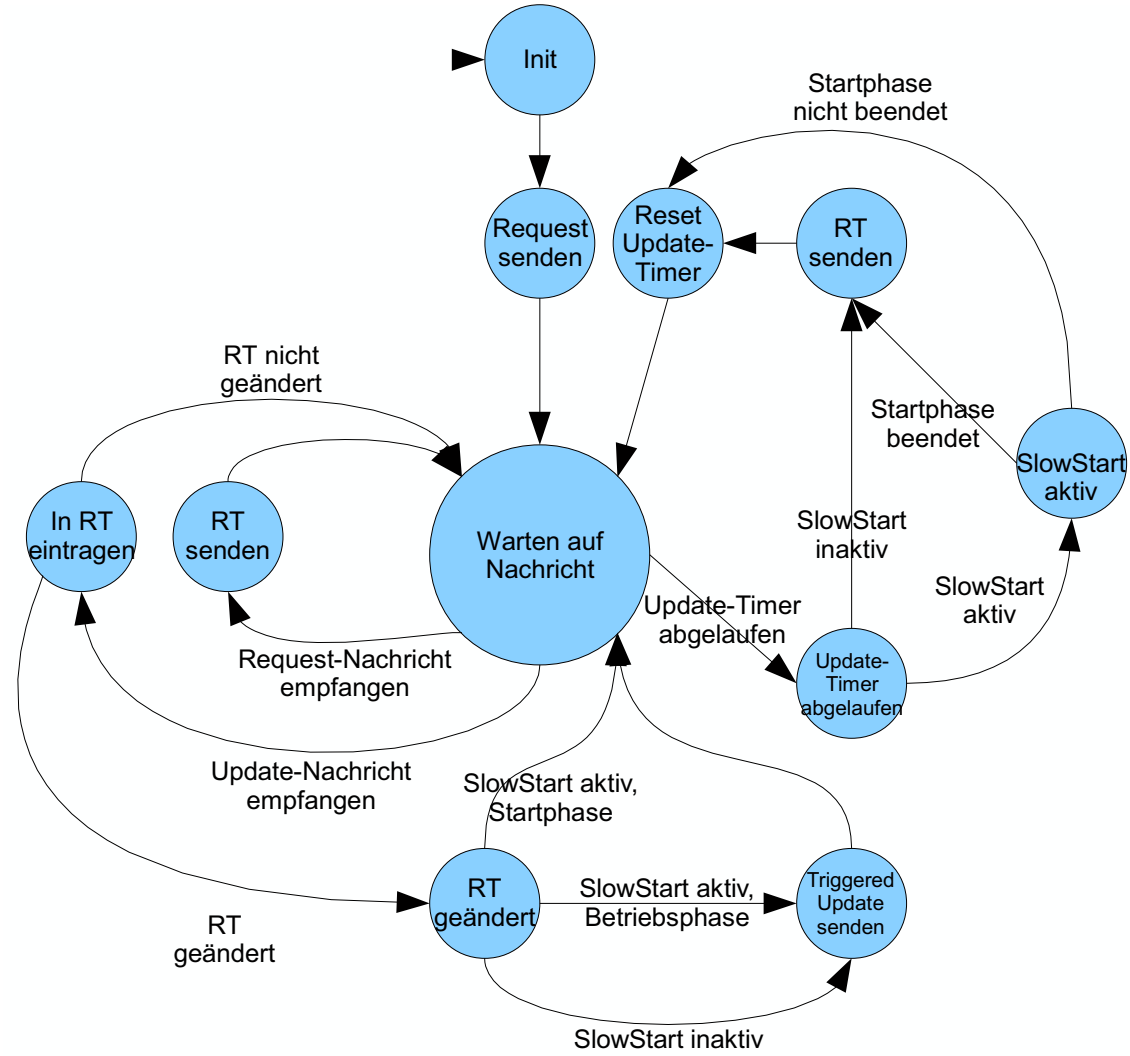


Abbildung 3.6: Ablauf von RIP mit integriertem Slow Start.

3.3.3 Vorteile

Slow Start bietet durch sein Verhalten in der Startphase zwei entscheidende Vorteile:

Robustere Schleifenerkennung: Der Router erhält beim Start von seinen Nachbarn deren momentan aktive Routingtabelle. Insbesondere für die Schleifenerkennung ist es hilfreich, die ursprüngliche Routingtabelle zu erhalten. Wird dem Router nämlich ein Netzwerk über verschiedene Interfaces gemeldet, kann er davon ausgehen, dass hier eine Schleife im Netzwerk vorhanden ist. Hätte der Router bereits Erreichbarkeitsinformationen versendet, könnte die eingehende Nachricht auch Informationen enthalten, welche der Router selbst verschickt hat und die bereits durch eine Schleife gelaufen sind. So könnte der Router eine Schleife zwischen zwei Interfaces erkennen, wo keine existiert. Dies ist bei Slow Start während der Startphase jedoch ausgeschlossen. Erhält der Router zu diesem Zeitpunkt Erreichbarkeitsinformationen zu einem Netzwerk über verschiedene Interfaces, so ist zwischen diesen Interfaces definitiv eine Schleife vorhan-

den. Die Erkennung von Schleifen wird also durch Slow Start einfacher und robuster. Dadurch ist RIP-MTI vollständig betriebsbereit, bevor der Router seinen eigentlichen Routingbetrieb aufnimmt.

Geringere Netzwerkbelastung: Beginnt der Router erst mit der Verbreitung seiner Routingtabelle, wenn er die Routingtabellen seiner Nachbarn erhalten hat, kann er mit einem Update direkt alle Netzwerke sowie zu jedem Netzwerk den besten Pfad propagieren. Das Netzwerk wird während der Startphase durch Slow Start also auf jeden Fall mit weniger Update-Nachrichten belastet. Dies gilt sowohl für RIP-MTI also auch für RIP.
Eine Messung, um welchen Faktor sich die Netzwerkbelastung durch Slow Start verringert, findet in Kapitel 4.3 statt.

3.3.4 Nachteile

Ganz ohne Nachteil kommt dieses Verfahren allerdings nicht aus. Durch das verzögerte Senden der Informationen dauert es natürlich nach dem Einschalten des Routers länger, bis das Netz wieder einen konvergenten Zustand erreicht hat. Eine schnelle Konvergenz ist in diesem Fall jedoch nicht zwingend erforderlich. Hierfür gibt es zwei Gründe:

- Geräte, welche ein anderes möglicherweise ausfallendes Gerät ersetzen sollen, werden normalerweise als so genannte „Hot-Spare“-Geräte integriert, so dass sie bei einem Ausfall des Hauptgerätes direkt einsatzbereit sind und nicht erst hochgefahren werden müssen. Gerne werden als Backup sogar Geräte unterschiedlicher Hersteller eingesetzt, um herstellerspezifischen Fehlern vorzubeugen (siehe Abbildung 3.7 auf Seite 42).
- Da der Router während der Startphase keine Update-Nachrichten verschickt, ist er auch keinem anderen Router im Netzwerk bekannt. Deshalb müssen von dem neuen Router noch keine Datenpakete vermittelt werden. Es kann also nicht zu Problemen im bestehenden Netzwerk kommen. Die Pakete werden weiterhin so geroutet, als ob der neue Router noch gar nicht vorhanden wäre. Es liegt somit kein echtes Konvergenzproblem vor.

Als größerer Nachteil entpuppt sich jedoch die Tatsache, dass Schleifen, welche nicht durch Netzwerk *d* erkannt werden konnten, nach der Startphase nicht mehr bestätigt werden können. Entsprechend läuft nach einiger Zeit der zugehörige Timer ab und löscht die Schleife aus der MSILM-Tabelle.
Möglicherweise könnte dies durch geschicktes Abfragen von Einträgen in den Routingtabellen der Nachbarrouter mittels Request-Nachrichten gelöst werden. Dies bedarf jedoch noch weiterer Erforschung.

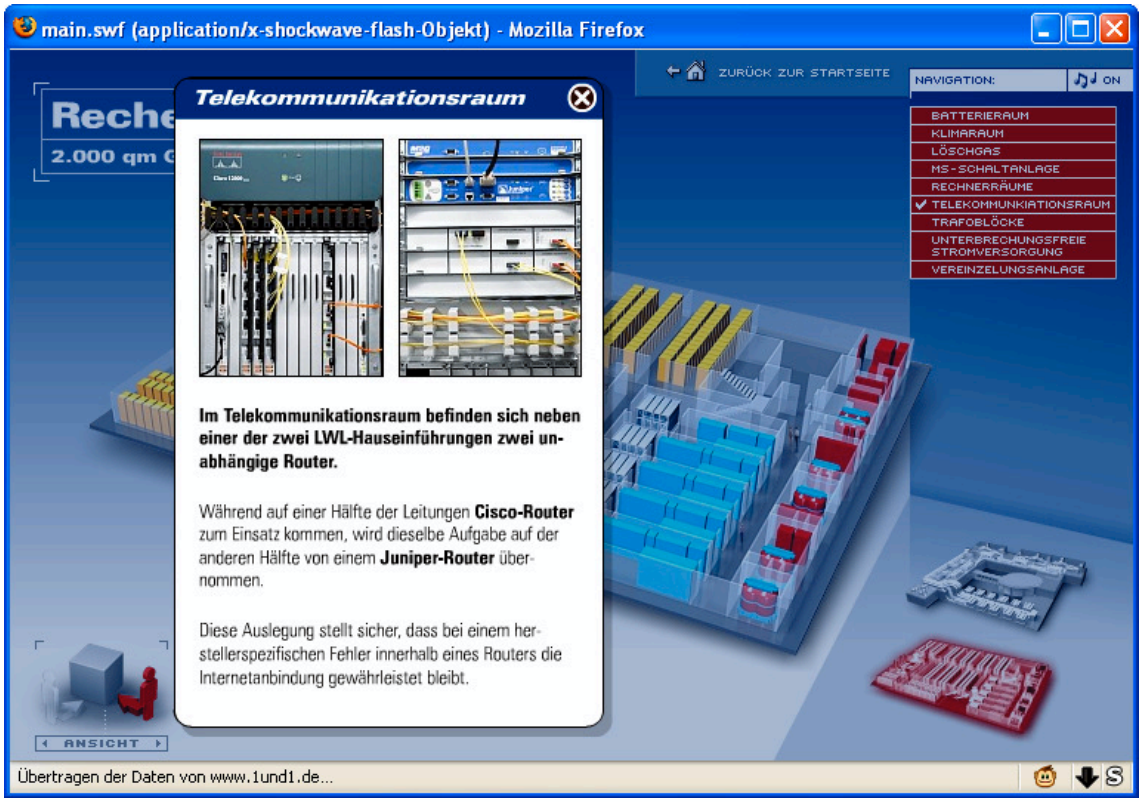


Abbildung 3.7: Redundante Anbindung des 1&1-Rechenzentrums
(Quelle: http://www.1und1.de/flash/main.swf)

3.3.5 Implementierung

In diesem Kapitel wird die Implementierung von Slow Start genau beschrieben. Als Grundlage diente die RIP-MTI Implementierung aus [BOH08], in welche die Slow Start Änderungen integriert wurden.

Abbildung 3.8 zeigt den Start des Routers. Wird der Router mit dem Startparameter `-s` gestartet, geht er in den Zustand „Slow Start enabled“ (Slow Start Erweiterung ist aktiviert). Ohne diesen Parameter startet der Router im Zustand „Slow Start disabled“ (Slow Start Erweiterung ist inaktiv, Router startet normal). Hierdurch kann Slow Start bei der Validierung einfacher mit dem normalen Start des Routers verglichen werden. Diese Modifikationen wurden in der Datei `rip_main.c` vorgenommen. Der Quelltext der Datei sowie eine ausführliche Dokumentation findet sich in Anhang E.4.

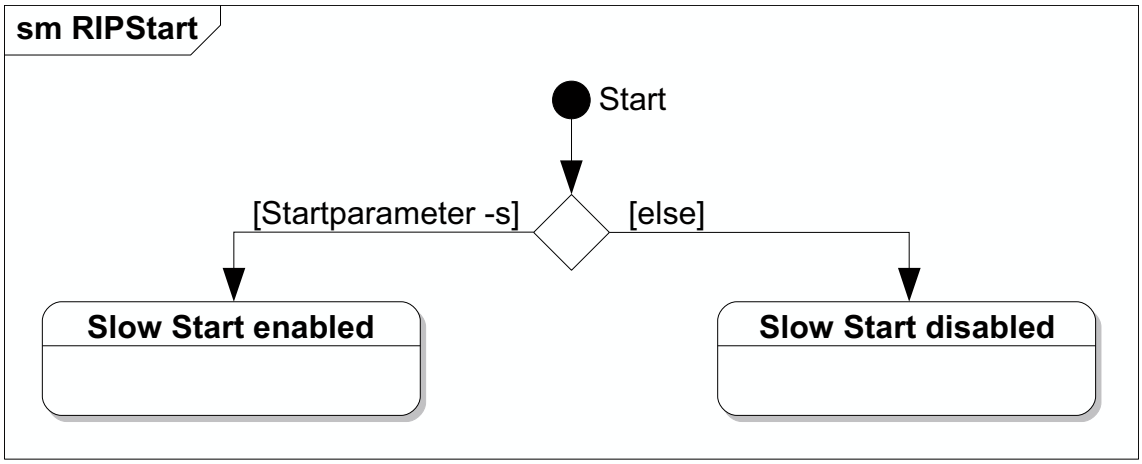


Abbildung 3.8: UML 2 Zustandsdiagramm: Starten des Routers

Wurde Slow Start aktiviert, befindet sich der Router im Zustand „Start“. Nach Ablauf des ersten Updateintervalls (25 bis 35 Sekunden) geht er in den Zustand „Running“ über, wie es im Timing-Diagramm in Abbildung 3.9 dargestellt ist.

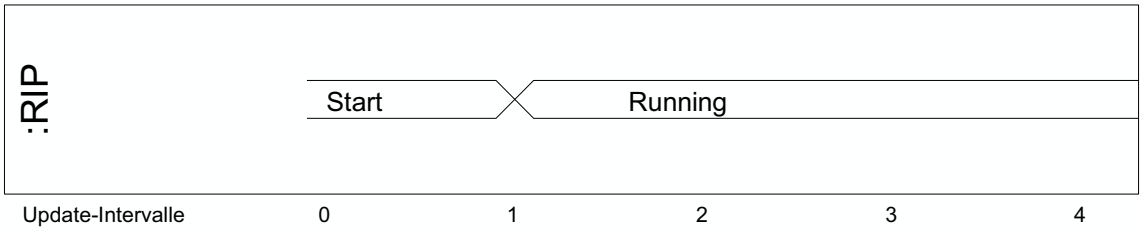


Abbildung 3.9: UML 2 Timing-Diagramm: Timing des Zustandsübergangs

Die Zeitspanne von einem Update-Intervall wurde gewählt, weil davon auszugehen ist, dass die Nachbarrouter das initiale Request des gestarteten Routers zügig beantworten und deshalb Verzögerungen von mehr als einem Updateintervall nicht notwendig sind. Ein Beenden der Startphase vor dem Ablauf eines Updateintervalls erscheint nicht sinnvoll. Hierbei könnte es passieren, dass der Router seine gesamte Routingtabelle per Triggered Update verschickt. Bei dem anschließenden Timed Update würden dann die gleichen Informationen innerhalb einer sehr kurzen Zeitspanne noch einmal übertragen werden.

Die Funktion für den Zustandswechsel wurde in die Datei `ripd.c` eingefügt und heißt `slowstart_modechange`. Der Quellcode dieser Datei ist in Anhang E.3 hinterlegt.

Im Zustand „Start“ versendet der Router lediglich die initialen Request-Nachrichten (Abbildung 3.10). Anschließend wartet er auf die Antworten seiner Nachbarrouter. Update-Nachrichten, insbesondere Triggered Updates, werden in diesem Zustand nicht ver-

sendet. Im anschließenden Zustand „Running“ arbeitet der Router genauso wie dies ohne Slow Start der Fall ist.

Die Unterdrückung der Update-Nachrichten im Zustand „Start“ wird ebenfalls in der Datei `ripd.c` vorgenommen. Sie findet in der Funktion `rip_triggered_update` statt.

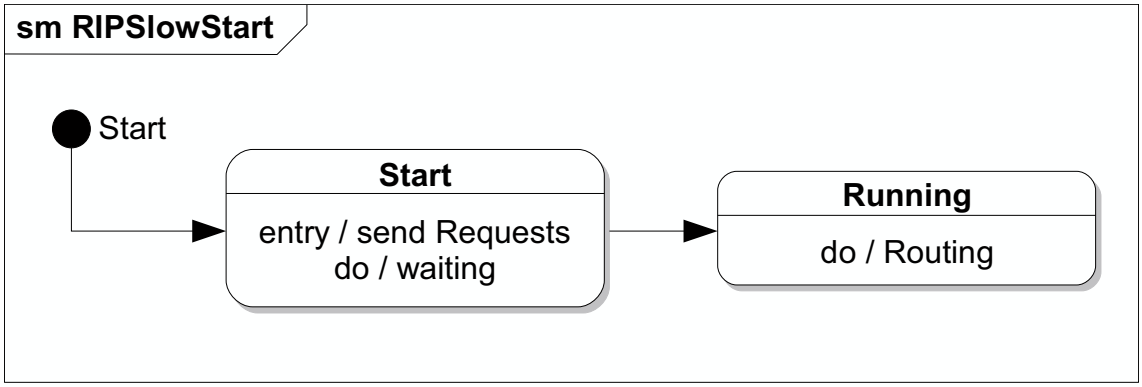


Abbildung 3.10: UML 2 Zustandsdiagramm: Zustände von Slow Start

Um die einzelnen Zustände zu speichern, wurde das globale Konstrukt `rip` erweitert. Es wird in der Datei `ripd.h` (siehe Anhang E.1) deklariert und speichert alle von RIP benötigten Parameter.

Wie in Abbildung 3.11 gezeigt, wurden drei Attribute zu dem Konstrukt hinzugefügt:

- `slow_start_state` speichert die in Abbildung 3.8 gezeigten Zustände. Das Attribut kann also entweder den Wert „Slow Start enabled“ oder den Wert „Slow Start disabled“ annehmen.
- `slow_start_mode` speichert die beiden Zustände aus Abbildung 3.10 und nimmt damit entweder den Wert „Start“ oder den Wert „Running“ an.
- `slow_start_updates` speichert, wie viele Updateintervalle die Startphase des Routers andauern soll. Nach Abbildung 3.9 ist dieses Attribut standardmäßig auf 1 gesetzt.

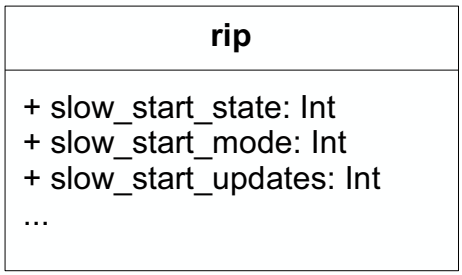


Abbildung 3.11: UML 2 Klassendiagramm: Erweiterung des globalen Konstrukts rip.

Wie in Kapitel 3.3.2 beschrieben, war ursprünglich eine unterschiedliche Behandlung einzelner Interfaces angedacht. Die hierfür benötigte Erweiterung in der Datei `rip_interface.c` wurde bereits vorgenommen. So wurde das Konstrukt `rip_interface` um das Attribut `slow_start_mode` erweitert (Abbildung 3.12), welches den Zustand des jeweiligen Interfaces speichert. Da diese Idee jedoch verworfen wurde, wird dieses Attribut aktuell nicht ausgewertet.

Der Quelltext dieser Datei ist in Anhang E.2 abgebildet.

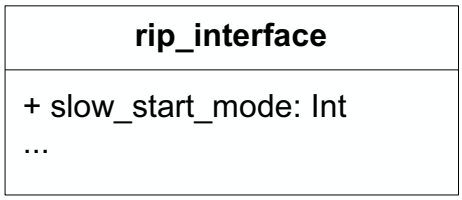


Abbildung 3.12: UML 2 Klassendiagramm zur Erweiterung des Konstrukts rip_interface.

Die oben beschriebenen Änderungen wurden auch an der aktuellen Version von RIP vorgenommen, da die Slow Start Erweiterung nicht nur Vorteile für RIP-MTI, sondern auch für RIP bringt. Hierzu wurde die quelloffene Routingsoftware Quagga[9] in der Version 0.99.11 verwendet. Prinzipiell wurden alle Erweiterungen genauso wie bei RIP-MTI beschrieben implementiert. Lediglich der in Abbildung 3.9 beschriebene Zustandsübergang konnte hier effizienter gelöst werden. So ist die zuvor beschriebene Funktion `slowstart_modechange` nicht notwendig. Der Zustandsübergang kann in der Funktion `rip_update`, welche für den Versand von Timed Updates zuständig ist, erledigt werden (Anhang E.5).

Somit ist die Implementierung von Slow Start vollständig. Im nächsten Kapitel wird das Verfahren nun validiert.

9 Quagga ist eine Routingsoftware unter der GPL. Unterstützte Protokolle sind OSPF, RIP und BGP. Die Software ist im Internet unter http://www.quagga.net/ erhältlich.

4 Validierung

Nach erfolgter Implementierung von Slow Start werden nun die in Kapitel 3 entwickelten Formeln zur Berechnung des Erwartungswertes der Zeitdauer vom Schließen der Schleife bis zu deren Erkennung, sowie der Quellcode von Slow Start validiert. Zuerst wird jedoch in Kapitel 4.1 die Testumgebung beschrieben, innerhalb welcher die Validierung stattfindet.

4.1 Testumgebung

Als Testumgebung wird VNUML benutzt, eine sehr reale Umgebung. Zur reproduzierbaren Steuerung der RIP-Daemons dient der später beschriebene RIP XTPeer.

4.1.1 VNUML

VNUML steht für „Virtual Network User Mode Linux“ und wurde 2002 am Telematics Engineering Department der Technischen Universität von Madrid entwickelt.[10] In diesem Kapitel wird der Umgang mit VNUML nur so weit beschrieben, wie es für das Verständnis der in Anhang F.1 abgebildeten Testsimulationen vonnöten ist. Ein ausführliches Tutorial zu VNUML findet sich in [MON07].

Wie der Name schon vermuten lässt, basiert VNUML auf „User Mode Linux“. In User Mode Linux wird unter dem eigentlichen Linux-Kernel ein weiterer Linux-Kernel als Anwendungsprozess gestartet, welcher eine virtuelle Maschine (VM) darstellt. In VNUML hingegen werden mehrere dieser Kernels gestartet. Jeder repräsentiert eine eigene VM. Diese VMs werden von VNUML zu einem virtuellen Netzwerk verbunden, in dem die Tests durchgeführt werden können.

Diese Vorgehensweise bietet mehrere Vorteile:

- Auf einem realen Computer kann ein ganzes Netzwerk mit vielen virtuellen Hosts simuliert werden, so dass die Hardwarekosten stark begrenzt sind.
- Jede virtuelle Maschine ist weitgehend unabhängig. Eine direkte Beeinflussung durch die anderen virtuellen Maschinen ist nicht möglich.

Die fehlende Berücksichtigung von Latenzzeiten bei VNUML wirkt sich bei den Tests nicht nachteilig aus, denn die Latenzzeit ist für den Test von Slow Start nicht relevant und kann bei Bedarf durch den später vorgestellten XTPeer simuliert werden. Bei den Formeln zur Berechnung der Zeitdauer für die Schleifenerkennung stellt die Latenzzeit

10 Die offizielle Webseite von VNUML ist unter http://jungla.dit.upm.es/~vnuml/ erreichbar. Unter http://www.uni-koblenz.de/~vnuml/ bietet die Universität Koblenz-Landau ebenfalls viele nützliche Informationen zu VNUML an.

lediglich eine Konstante dar, wodurch sich deren Fehlen nicht auf die Validierung auswirkt sollte.

VNUML besteht aus zwei wesentlichen Komponenten: der XML-Datei (Extensible Markup Language) und dem Perl-Skript.
Die zu erstellende XML-Datei enthält die Spezifikationen der virtuellen Maschinen sowie der Netzwerke für die aktuell gewünschte Simulation und ist folgendermaßen aufgebaut: Root-Element ist das XML-Tag `<vnuml>`. Unterhalb dieses Elementes werden zuerst die globalen Einstellungen mit dem Tag `<global>` spezifiziert. Hier wird unter anderem die VNUML-Versionsnummer (in dieser Arbeit wurde VNUML in der Version 1.8 eingesetzt) und der Simulationsname eingetragen. Durch das Tag `<vm_mgmt>` kann für jede VM ein Management Interface definiert werden. Über dieses Interface können Daten zwischen der VM und dem Hostsystem ausgetauscht werden. Dies ist insbesondere für den im folgenden Kapitel beschriebenen XTPeer wichtig. Nach den globalen Spezifikationen werden die einzelnen Netzwerke definiert. Dies geschieht über das Tag `<net>`. Anschließend erfolgt die genaue Spezifizierung der einzelnen Hosts über das Tag `<vm>`. Für jeden Host (bzw. jede virtuelle Maschine) müssen die Netzwerkinterfaces und die dazu gehörenden Adressen festgelegt werden.

Nachdem alle gewünschten Netzwerke und Hosts in der XML-Datei spezifiziert wurden, muss die Simulation mit dem Perl-Skript `vnumlparser.pl` – der zweiten wichtigen Komponente von VNUML – geparst werden. Gestartet wird sie dann über den Befehl `vnumlparser.pl -t simulation.xml -v -u root`.

Läuft die Simulation vollständig, kann die benötigte Software (hier mindestens Quagga mit dem RIP-Daemon bzw. RIP-MTI-Daemon) gestartet werden. In der XML-Datei können mit dem Tag `<exec>` zu jeder VM Exec-Sequenzen angelegt werden, damit die Software nicht auf jeder Maschine manuell gestartet werden muss. Die Exec-Sequenzen werden über folgenden Befehl ausgeführt:

```
vnumlparser.pl -x sequenzname@simulation.xml -v -u root
```

Der Befehl `vnumlparser.pl -d simulation.xml -v -u root` beendet die Simulation.

4.1.2 RIP XTPeer

Bei dem RIP XTPeer (XT steht für „extern triggerbar“) handelt es sich um eine an der Universität Koblenz-Landau von Stefan Lange [LAN07] entwickelte Erweiterung für RIP-MTI. Mit dem XTPeer wird das Updateverhalten des RIP-MTI-Daemons steuerbar. Bestimmte Situationen wie das Auftreten eines CTIs sind so unter kontrollierten Bedingungen reproduzierbar.

Lange fügte dem RIP-MTI-Daemon einen sogenannten XT_Server sowie einen SL_Client hinzu.

XT_Server: Über den XT_Server ist es möglich von außen mit einem XT_Client eine Verbindung zu RIP-MTI aufzubauen und den Daemon verschiedene Kommandos ausführen zu lassen. So kann zum Beispiel auf einem bestimmten Interface eine Update-Nachricht ausgelöst oder auch unterdrückt werden.

SL_Client: Der SL_Client kann sich mit einem externen SL_Server verbinden und diesem Informationen zum aktuellen Status von RIP-MTI senden. Hierbei wird unter anderem die Routingtabelle an den Server übermittelt.

Der eigentliche XTPeer besteht aus dem XT_Client sowie dem SL_Server, den Gegenstücken der zuvor beschriebenen Komponenten XT_Server und SL_Client des RIP-MTI-Daemons. Er läuft auf dem Hostsystem und verbindet sich über die Management Interfaces der einzelnen VMs mit den darauf laufenden RIP-Daemons bzw. erhält von dort seine Daten. Der XTPeer bietet eine graphische Oberfläche. In dieser werden die vom SL_Server empfangenen Informationen der einzelnen Router dargestellt. Sie können nun über den XT_Client gesteuert werden. Hierzu werden im XTPeer die einzelnen Router mit ihren Interfaces sowie die Netzwerkverbindungen zwischen den Interfaces graphisch dargestellt. Neben verschiedenen Einstellungen für RIP-MTI (z.B. Auswahl, ob Normal Mode, Strict Mode oder einer der Careful Modes benutzt werden soll) können auf jedem Interface manuell Update-Nachrichten getriggert werden (Modus MANUAL). Der Modus AUTOTRIGGER verschickt zusätzlich zu den manuell möglichen Updates bei Änderungen an der Routingtabelle Triggered Updates. Im Modus AUTO werden automatisch Triggered Updates sowie regelmäßige Timed Updates auf dem jeweiligen Interface verschickt, außerdem sind ebenfalls manuelle Updates möglich.

4.2 Validierung der Formeln für die zu erwartende Zeitdauer bis zur Erkennung einer Schleife in RIP-MTI

Nachdem jetzt die Rahmenbedingungen für die Testphase klar sind, gilt es die in Kapitel 3.1.4 entwickelten Formeln 3.12 und 3.13 für die Erwartungswerte zur Berechnung der Zeitdauer der Schleifenerkennung experimentell zu überprüfen. Die Minimal- und Maximalwerte sowie die entwickelten Teilformeln sind nur theoretischer Art und werden daher hier nicht validiert. Folgend zuerst die Beschreibung des Testverfahrens, anschließend die Auswertung.

4.2.1 Testverfahren

Da für die Erkennung einer topologischen Schleife die Erreichbarkeitsinformationen zu Netzwerk *d* lediglich über mehrere Router hinweg verbreitet werden müssen, wird zum Testen jeweils eine sehr einfache Netzwerktopologie verwendet. In allen Versuchen wird die Zeitspanne gemessen, bis Netzwerk *d* Router *R1* bekannt ist. Je nach zu prüfender Formel unterscheiden sich die Tests geringfügig:

Szenario 1: In Szenario 1 wird die Schleife durch einen neuen Router geschlossen. Zur Berechnung der zu erwartenden Zeitdauer bis zur Erkennung der Schleife durch Router *R1* wurde Formel 3.12 entwickelt. Um die Richtigkeit der Formel experimentell zu prüfen, wird die in Abbildung 4.1 gezeigte VNUML-Simulation verwendet. Die hierzu gehörende XML-Datei ist – wie die des folgenden Szenarios – in Anhang F.1 abgebildet.

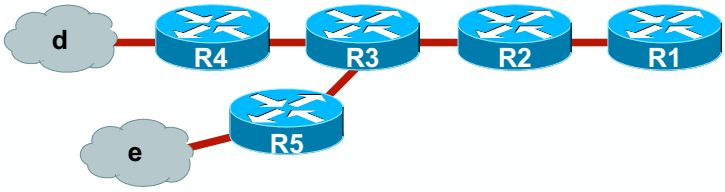


Abbildung 4.1: Netzwerktopologie zur Validierung der Zeitdauer bei Szenario 1.

R4 stellt den in Kapitel 3 definierten Router R_d dar. Sein Interface zu *R3* wird im XT-Peer auf MANUAL gestellt, so dass die Verfügbarkeit von Netzwerk *d* nicht weiterverbreitet wird. Zum Zeitpunkt $t=0$ wird an diesem Interface ein Update getriggert. Es simuliert das Response auf das Request von Router *R3*, durch den die Schleife geschlossen wurde. Das Interface von *R3* zu *R2* wird auf den Modus AUTOTRIGGER gestellt. So werden eingehende Routen per Triggered Update an *R2* weiter gemeldet. Router *R5* dient der Verzögerung des Triggered Updates an *R3*. Sein Interface zu *R3* steht auf MANUAL. Bevor an R3 die Route zu Netzwerk d eintrifft, wird von *R5* ein Update mit der Erreichbarkeit von Netzwerk *e* an *R3* geschickt. Dieses wird von *R3* per Triggered Update weiterverbreitet und der Timer wird gestartet. Die Erreichbarkeitsinformation von Netzwerk *d* wird also verzögert. Das Interface von *R2* zu *R1* wird ebenfalls auf MANUAL gestellt. Hierdurch wird der Verlust der Update-Nachricht simuliert. Sobald *R2* die Route zu Netzwerk *d* bekannt ist, wird das Interface zu *R1* auf den Modus AUTO gestellt. So werden die Informationen zur Erreichbarkeit von Netzwerk d mit dem nächsten Timed Update an *R1* übertragen. Sind diese Informationen bei *R1* eingetroffen, wird die Zeitmessung gestoppt.

Szenario 2: Bei Szenario 2 werden zwei Router durch ein neues Netzwerk verbunden, wodurch eine Schleife entsteht. Formel 3.13 wurde zur Berechnung der zu erwartenden Zeitdauer vom Schließen der Schleife bis zur deren Erkennung durch Router *R1* entwickelt.

In Abbildung 4.2 ist die VNUML-Simulation abgebildet, mit der die Korrektheit von Formel 3.13 überprüft werden soll.

Abbildung 4.2: Netzwerktopologie zur Validierung der Zeitdauer bei Szenario 2.

Auch hier stellt *R4* wieder den Router R_d dar. Wie zuvor wird das Interface von *R4* zu *R3* auf den Modus MANUAL gestellt. Allerdings wird nun zum Zeitpunkt $t=0$ keine Update-Nachricht getriggert, sondern das Interface wird auf den Modus AUTO umgestellt. So wird das Schließen der Schleife durch das Netzwerk zwischen *R4* und *R3* simuliert, bei dem die Erreichbarkeitsinformationen zu Netzwerk *d* mit dem nächsten Timed Update übertragen werden. Die Interfaces von *R3* zu *R2* sowie von *R2* zu *R1* sind auf AUTOTRIGGER gestellt. So wird die Erreichbarkeit von Netzwerk *d* per Triggered Updates weiterverbreitet. Ist Router *R1* Netzwerk *d* bekannt, wird die Zeitmessung gestoppt.

Jeder einzelne der beiden gerade erläuterten Versuche muss möglichst oft durchgeführt werden, damit der errechnete Erwartungswert bestätigt werden kann.[11] Aus den so gewonnenen Daten wird jeweils der Mittelwert gebildet und mit dem entsprechenden errechneten Erwartungswert verglichen. Die Ergebnisse zeigt das folgende Kapitel „Testauswertung“.

4.2.2 Testauswertung

Am Beginn der Auswertungsphase steht die Berechnung der Erwartungswerte mit den durch das Testverfahren gegebenen Zahlen nach den in Kapitel 3.1.4 vorgestellten zugehörigen Formeln. Zu jedem der beiden Tests wird ein Diagramm erstellt, in dem der berechnete Wert mit den gemessenen Werten verglichen wird. Auf der y-Achse ist jeweils die Zeitdauer zur Übermittlung der Routeninformationen von Netzwerk *d* an Router *R1* abgetragen, auf der x-Achse die Anzahl der durchgeführten Versuche. Die rote Kurve zeigt dabei stets den Durchschnittswert der gemessenen Zeitspannen aus den bislang durchgeführten Versuchen. Der errechnete Erwartungswert ist in hellblau dargestellt. Stimmt die Formel, so nähert sich die rote Kurve mit steigender Anzahl an Versuchen der hellblauen Geraden immer weiter an.
Die Messergebnisse jedes einzelnen Tests sind in Anhang F.2 zu finden.

Szenario 1: Bei der Berechnung des Erwartungswertes ist wegen der Verwendung von VNUML die Latenzzeit mit 0 Sekunden anzunehmen. Die Anzahl c_R der Router, an

11 Zwischen den einzelnen Versuchen müssen die Routingtabellen der einzelnen Router gelöscht werden, damit für jeden Testdurchlauf die gleichen Ausgangsbedingungen gelten.

denen Verzögerungen auftreten können, ist gleich drei. An einem dieser drei Router findet eine Verzögerung durch den Timer statt, daher ist für p_t 1/3 zu wählen. In dem Netzwerk zwischen *R2* und *R1* wird der Paketverlust simuliert, so dass p_v hier gleich 0,5 ist (das Triggered Update geht verloren, das nächste Timed Update überträgt die Informationen). In den restlichen Netzwerken ist p_v gleich null. Nach Einsetzen der Werte in Formel 3.12 und Berechnung der Teilergebnisse ergibt sich eine gesamte zu erwartende Verzögerung von 605/36 (16,8) Sekunden (Formel 4.1).

Verzögerung in den Netzwerken ohne Paketverluste:

$$t_{v_{2,3}} = \left(\frac{1}{1-0} - 1\right) \cdot \frac{545}{36}\,\text{sec.} = 0\,\text{sec.}$$

Verzögerung in dem Netzwerk mit Paketverlust:

$$t_{v_1} = \left(\frac{1}{1-\frac{1}{2}} - 1\right) \cdot 30\,\text{sec.} - \frac{535}{36}\,\text{sec.} = \frac{545}{36}\,\text{sec.} \tag{4.1}$$

$$t_\mu = 0\,\text{sec.} + \sum_{R=1}^{3} \frac{1}{3} \cdot \frac{5}{3}\,\text{sec.} + t_{v_R} = \frac{605}{36}\,\text{sec.}$$

In Abbildung 4.3 wird nun der gerade berechnete Erwartungswert mit den Ergebnissen der Versuche verglichen. Wie zu sehen ist, nähert sich der Durchschnitt der Messwerte dem Erwartungswert an. Im Bereich von Versuch 12 bis 18 stimmen beide sogar fast überein. Anschließend wird der gemessene Durchschnittswert wieder geringfügig größer als der Erwartungswert. Nach 34 Versuchen unterschieden sich beide um 2,9 Sekun-

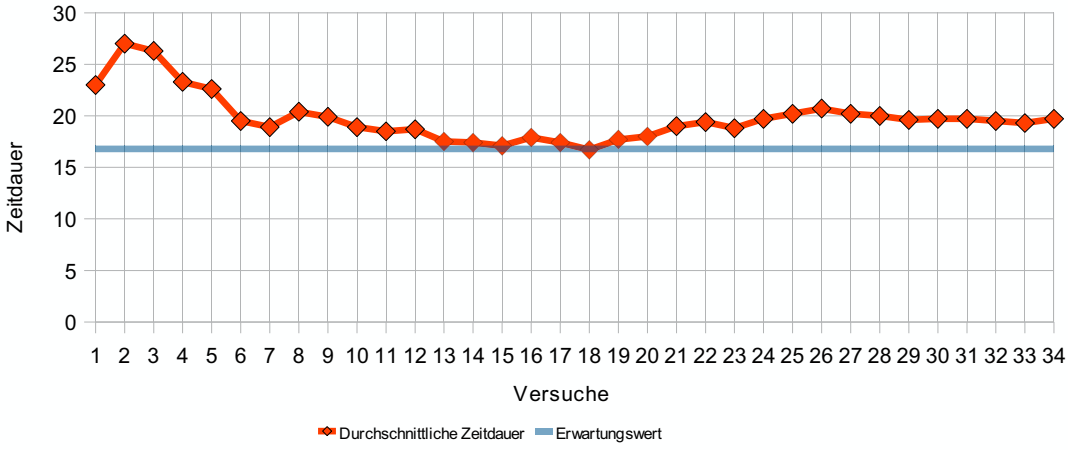


Abbildung 4.3: Szenario 1: Vergleich berechneter Erwartungswert – Messergebnisse.

den, was bei einer möglichen Zeitspanne von mehreren Minuten zwischen der minimalen und maximalen Laufzeit nicht besonders viel ist.

Szenario 2: Wieder erfolgt zuerst die Berechnung: Nach Formel 3.13 werden zuerst die Teilverzögerungen berechnet. Hierbei werden zuerst die Verzögerungen durch Paketverluste berechnet. Da jedoch entsprechend der Versuchsbeschreibung keine Paketverluste auftreten, sind diese Verzögerungen null. c_R muss auf den Wert 3 gesetzt werden, da sich an drei Routern Verzögerungen ergeben können. p_t ist nach den Vorgaben des Tests null. Somit ergibt sich nach Formel 4.2 eine erwartete Verzögerung von 545/36 (15,1) Sekunden.

Verzögerung bei Verlust des ersten Timed Updates:

$$t_{v_1} = \left(\frac{1}{1-0} - 1 \right) \cdot 30\,\text{sec.} = 0\ \text{sec.}$$

Verzögerung bei Verlust der Triggered Updates:

$$t_{v_R} = \left(\frac{1}{1-0} - 1 \right) \cdot \frac{545}{36}\,\text{sec.} = 0\,\text{sec.} \tag{4.2}$$

Berechnung des Erwartungswertes:

$$t_\mu = 0\,\text{sec.} + \frac{545}{36}\,\text{sec.} + t_{v_1} + \sum_{R=2}^{3} 0 \cdot \frac{5}{3}\,\text{sec.} + t_{v_R} = \frac{545}{36}\,\text{sec.}$$

Ein Vergleich des berechneten Erwartungswertes mit den bei den Tests gemessenen Verzögerungen in Abbildung 4.4 zeigt, dass sich der Durchschnittswert der Messergebnisse dem berechneten Erwartungswert immer weiter nähert.

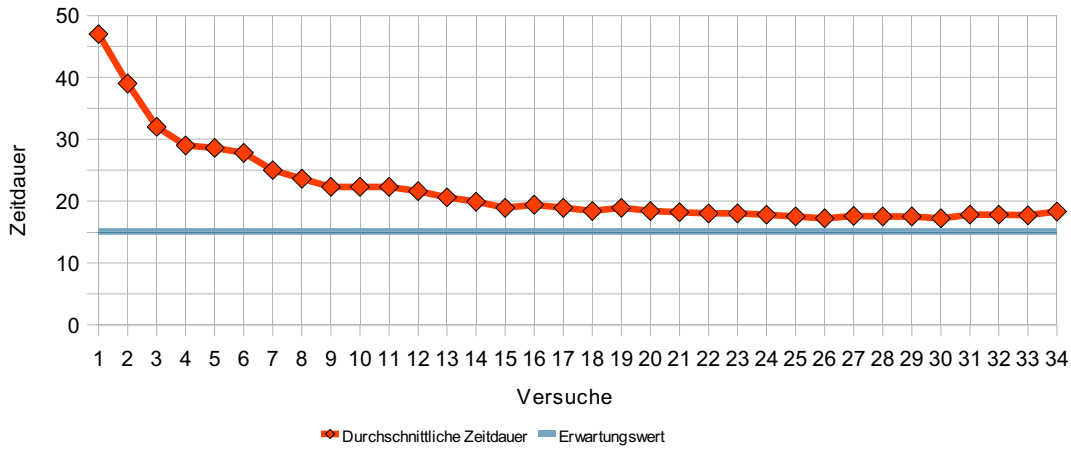


Abbildung 4.4: Szenario 2: Vergleich des berechneten Erwartungswertes mit den Messergebnissen.

Nach 34 Versuchen weicht der Durchschnittswert nur noch um 3,2 Sekunden vom Erwartungswert ab. Diese Abweichung könnte möglicherweise an einem Messfehler bei der ersten Versuchsdurchführung liegen. Hierbei wurde eine Zeitspanne von 47 Sekunden ermittelt, welche erheblich von den restlichen Messergebnissen abweicht. Wird dieser erste Messwert nicht berücksichtigt beträgt die Differenz von Durchschnitts- und Erwartungswert nur noch 2,2 Sekunden. Dies entspricht jeweils auch der Standardabweichung.

Bei der Betrachtung beider Testreihen fällt auf, dass jeweils der Durchschnittswert geringfügig größer ist als der Erwartungswert. Da es sich bei beiden Testreihen um einen ähnlichen Wert (2,9 Sekunden gegenüber 3,2 Sekunden) handelt, könnte unter Umständen eine doch spürbare Latenzzeit von VNUML vorliegen. Auch Ungenauigkeiten bei der Zeitmessung müssen in Betracht gezogen werden, denn in den Logdateien wird die Zeit nur in ganzen Sekunden angegeben. Weitere Ungenauigkeiten können auftreten, da der XTPeer manuell bedient wird. So ist es beispielsweise möglich, dass durch das manuelle Umstellen des Interfaces an *R2* im ersten Szenario bei dicht aufeinander folgenden Triggered und Timed Updates nicht nur das Triggered, sondern auch das folgende Timed Update unterdrückt wurde. Somit läge die Verlustrate in diesem Versuch statt bei den angenommenen 0,5 bei 0,66.

Insgesamt ist mit den beiden Formeln jedoch eine recht genaue Abschätzung der Zeitdauer vom Schließen einer Schleife bis zu deren Erkennung möglich.

4.3 Validierung von Slow Start

Slow Start bietet zwei Vorteile: Eine robustere Schleifenerkennung für RIP-MTI sowie eine effizientere Konvergenz nach dem Start des Routers. Letzteres ist der entscheidende Vorteil von Slow Start und wird in diesem Kapitel getestet.

4.3.1 Testverfahren

Die Verringerung der Anzahl verschickter Update-Nachrichten durch Slow Start in der Startphase wird mithilfe von zwei verschiedenen Szenarien getestet:

1. Im ersten Testfall werden mit dem neuen Router zwei komplett getrennte Netzwerke miteinander verbunden. Durch den neuen Router wird keine Schleife geschlossen und den Nachbarroutern sind keine Routen zum jeweils anderen Netzwerk bekannt.

   In Abbildung 4.5 ist die Netzwerktopologie für diese VNUML-Simulation dargestellt. Die zugehörige XML-Datei ist – genauso wie die des folgenden Testfalls – in Anhang F.1 zu finden.

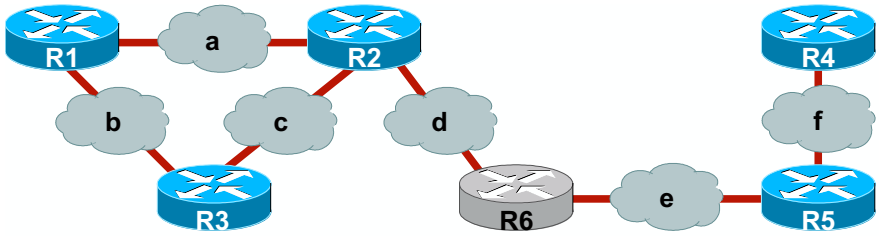


Abbildung 4.5: Topologie der ersten Testsimulation für Slow Start.

R6 bezeichnet in der Simulation den neuen Router, der die beiden Netzwerke miteinander verbindet. Um die Funktion von Slow Start zu testen, wird *R6* einmal mit und einmal ohne Slow Start gestartet. Über Wireshark wird jeweils der Nachrichtenversand durch RIP in den einzelnen Teilnetzen mitverfolgt.

Da in diesem Szenario Router *R6* keiner Schleife angehört, ergeben sich für RIP-MTI keinerlei Vorteile in der Schleifenerkennung. Es wird lediglich ein besseres Konvergenzverhalten nach dem Start des Routers erwartet.

2. Beim zweiten Testfall wird der Router in ein bestehendes Netzwerk integriert. Hierbei wird auf jeden Fall mindestens eine neue topologische Schleife im Netzwerk erzeugt. Den Nachbarroutern sind bereits alle Netzwerke bekannt. Durch den neuen Router erhalten sie jedoch zu einigen Teilnetzen kürzere Routen.

   Abbildung 4.6 zeigt die in dieser Testsimulation verwendete Netzwerktopologie. Wie im vorhergehenden Testfall wird Router *R6* einmal mit und einmal ohne Slow Start gestartet. Die Verfolgung der verschickten Nachrichten erfolgt wieder über Wireshark.

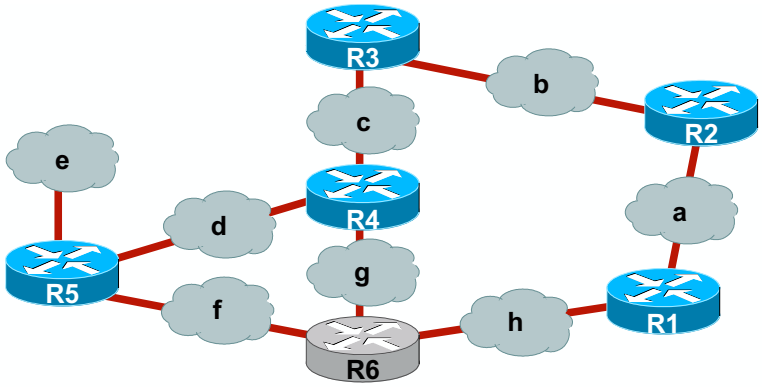


Abbildung 4.6: Topologie der zweiten Testsimulation für Slow Start.

4.3.2 Testauswertung

Die einzelnen Test wurden wie im vorherigen Kapitel beschrieben durchgeführt. Dabei konnten interessante Daten gewonnen werden, die folgend ausgewertet werden.
Da die gesamten Testdaten sehr umfangreich sind, sich jedoch sehr ähneln, wurden die

in diesem Kapitel vorgestellten Resultate in Anhang F.3 nur mit den wichtigsten Testdaten belegt. Die vollständigen Testdaten sind auf der beiliegenden CD-ROM zu finden.

Testfall 1: Zuerst erfolgt der Start von Router *R6* ohne Slow Start. Nach dem Einschalten verschickt *R6* über seine beiden Interfaces Request-Nachrichten an *R2* und *R5*. Erstaunlich ist hierbei, dass R6 direkt nach Versand des ersten Requests auf dem selben Interface eine Update-Nachricht mit den ihm bekannten Netzwerken *d* und *e* verschickt. Hierdurch werden die entsprechenden Einträge in der Routingtabelle nicht mehr als „geändert“ geführt, was einen Versand von Triggered Updates mit diesen Informationen auf den restlichen Interfaces – nach dortigem Versand der initialen Requests – verhindert. Antworten *R2* und *R5* auf die Request-Nachrichten, schickt *R6* nach Ablauf des Timers des vorausgegangenen Triggered Updates seinen Nachbarroutern die neu gewonnenen Informationen. Erst mit dem ersten Timed Update verschickt der Router seine vollständige Routingtabelle inklusive der direkt angeschlossenen Netzwerke an alle Nachbarrouter. In Abbildung 4.7 ist links die Kommunikation zwischen *R6* und *R5* über Netzwerk *e* dargestellt. *R5* steht hierbei exemplarisch für die Nachbarrouter von *R6*, deren Kommunikation ähnlich abläuft.

Bei den Tests mit VNUML wurde das initiale Triggered Update mit den direkt angeschlossenen Netzwerken regelmäßig über Interface 1, dem Management Interface der VM, verschickt. Hierdurch wurden sowohl in Netzwerk *d* als auch in Netzwerk *e* die Erreichbarkeit des jeweils anderen Netzwerkes erst mit dem ersten Timed Update übertragen. Entsprechend ist das initiale Triggered Update nicht in Abbildung 4.7 zu finden.

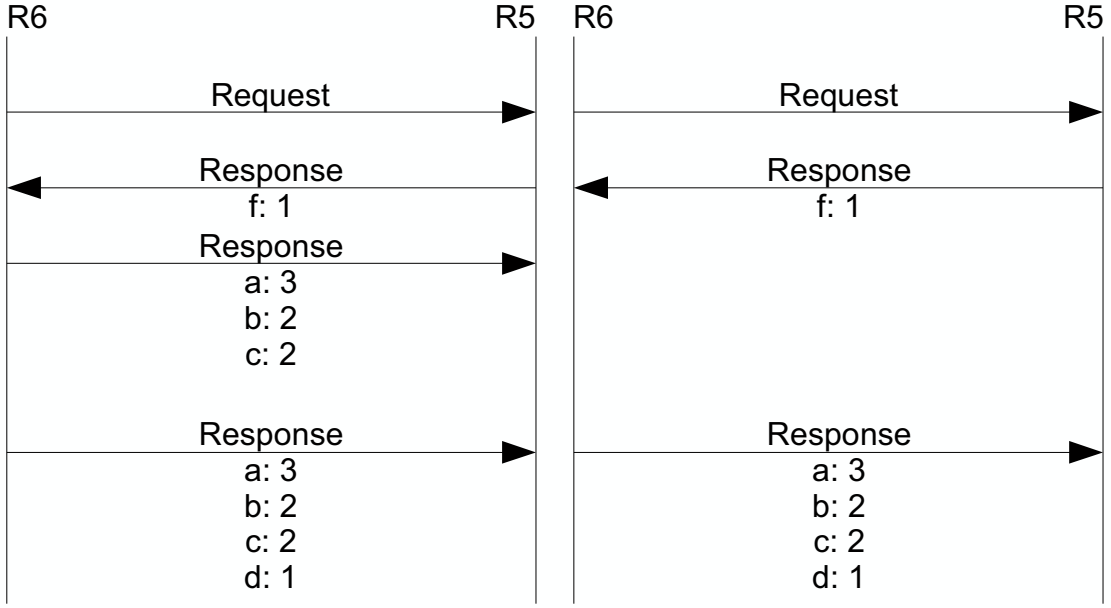


Abbildung 4.7: Vergleich des Nachrichtenversandes zwischen R5 und R6 über Netzwerk e. Links ohne Slow Start, rechts mit Slow Start.

Mit aktiviertem Slow Start läuft der Start von Router *R6* folgendermaßen ab: Wie zuvor verschickt *R6* nach dem Einschalten über seine beiden Interfaces Request-Nachrichten an seine beiden Nachbarrouter *R2* und *R5*. Nun wartet *R6* auf Antwort, ehe er weitere Nachrichten verschickt. *R2* und *R5* beantworten die Request-Nachricht von *R6*. *R6* besitzt nun eine vollständige Routingtabelle. Nachdem der Update Timer zum ersten Mal abgelaufen ist, verschickt *R6* seine Routingtabelle an seine beiden Nachbarrouter. Die Kommunikation zwischen *R6* und *R5* über Netzwerk *e* bei aktiviertem Slow Start ist in Abbildung 4.7 rechts dargestellt. Screenshots des Nachrichtenverkehrs mit und ohne Slow Start in Wireshark sind in Anhang F.3 zu finden.

Es ist festzustellen, dass durch Slow Start in Netzwerk *e* eine Nachricht eingespart wird. In Netzwerk *d* sind die Beobachtungen entsprechend. Für die anderen Teilnetze bedeutet dies, dass hier statt zwei Triggered Updates jeweils nur eines verschickt wird, bis das gesamte Netzwerk konvergent ist. Im gesamten restlichen Netzwerk werden in diesem Szenario mit Slow Start also bis zur Konvergenz 50 % der Update-Nachrichten gegenüber einem Start ohne Slow Start eingespart.

Durch den unkonventionellen Versand des initialen Triggered Updates bei deaktiviertem Slow Start ist die Konvergenzzeit identisch zu der Konvergenzzeit beim Start des Routers mit Slow Start.

Testfall 2: Wie beim vorherigen Testfall erfolgt der Start von Router *R6* zuerst ohne Slow Start. Die Kommunikation von *R6* und *R1* über Netzwerk *h* nach dem Start von *R6* bis zur Konvergenz des gesamten Netzwerkes sieht dabei folgendermaßen aus: *R6* verschickt eine Request-Nachricht an *R1*. Die Erreichbarkeit der direkt angeschlossenen Netzwerke teilt *R6* wieder nur über das Management Interface (erstes Interface) mit.

Je nachdem, in welcher Reihenfolge und in welchen zeitlichen Abständen *R4* und *R5* auf das Request von *R6* antworten, verschickt *R6* ein oder zwei weitere Triggered Updates an *R1*, bis das Netzwerk konvergent ist: Trifft das Response von *R4* vor dem von *R5* ein, so wird Netzwerk *e* einmal mit der Metrik 3 und direkt anschließend mit der Metrik 2 an *R1* gemeldet.
Die Übermittlung der direkt angeschlossenen Netzwerke *f* und *g* erfolgt mit dem ersten Timed Update.

Ist Slow Start beim Start von *R6* aktiviert, gestaltet sich die Kommunikation zwischen *R6* und *R1* so: *R6* sendet das initiale Request an *R1* und *R1* beantwortet dieses mit seiner vollständigen Routingtabelle. Nach Ablauf des ersten Updateintervalls sendet *R6* seine vollständige Routingtabelle an *R1*.

In Abbildung 4.8 ist der Vergleich wieder graphisch dargestellt. Links die Nachrichtenübermittlung ohne Slow Start, rechts mit Slow Start. Mit Slow Start werden nach dem Start von Router *R6* in Netzwerk *h* bis zu zwei Nachrichten eingespart, bis das Netz-

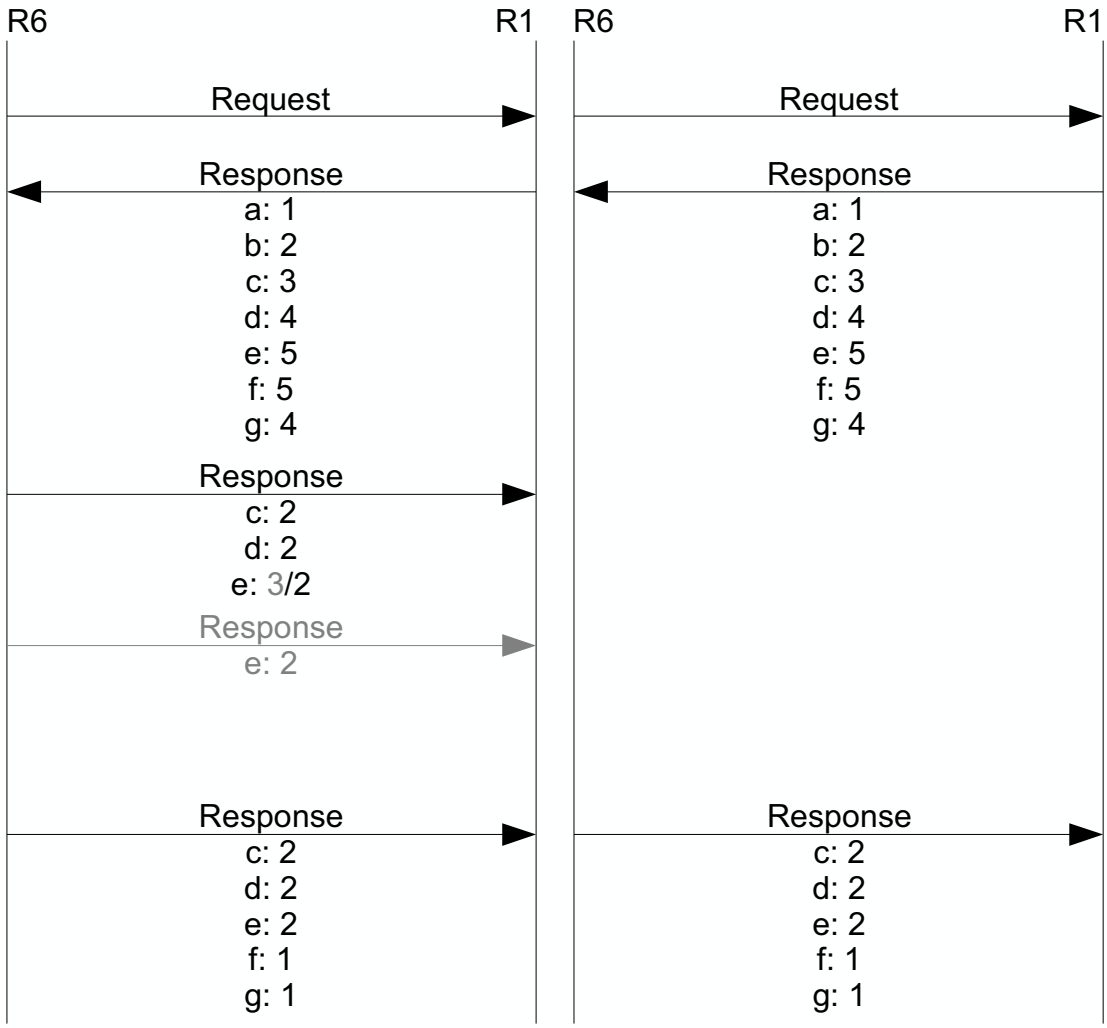


Abbildung 4.8: Vergleich des Nachrichtenversandes zwischen R1 und R6 über Netzwerk h. Links ohne Slow Start, rechts mit Slow Start.

werk konvergent ist. Im gesamten Netzwerk werden bei diesem Testfall durch Slow Start wieder ungefähr 50 % der Update-Nachrichten zwischen den Routern eingespart.

Diese Einsparmöglichkeiten sind sehr wahrscheinlich in allen Netzwerktopologien realistisch. Mit Slow Start ist die Anzahl der verschickten Update-Nachrichten in den direkt an den neu gestarteten Router angeschlossenen Netzwerken stets gleich 2, wobei immer eine Update-Nachricht von dem Nachbarrouter des neu gestarteten Routers stammt und eine an den Nachbarrouter verschickt wird. Dieser verbreitet also lediglich eine Update-Nachricht über das restliche Netzwerk weiter.

Ohne Slow Start liegt die Anzahl der verschickten Update-Nachrichten in den an den neu gestarteten Router angrenzenden Netzwerken bei mindestens 3. Hiervon ist eine Update-Nachricht an den neu gestarteten Router gerichtet. Die übrigen (mindestens 2) Update-Nachrichten werden von dem Nachbarrouter über das rechtliche Netzwerk weitergeleitet.

Theoretisch können zwar bei einem Router mit vielen Interfaces auch entsprechend viele „bessere“ Routen zu einem Netzwerk existieren, die nacheinander über das Netzwerk verbreitet werden, da aber wegen des Timers Triggered Updates zusammengefasst werden, sollte die Anzahl der über das gesamte Netzwerk zu verbreitenden Update-Nachrichten nicht wesentlich größer als zwei sein.

Somit können durch Slow Start im Allgemeinen etwa 50% der Update-Nachrichten eingespart werden. Höhere Raten sind jedoch durchaus denkbar.

5 Fazit

In Kapitel 3 wurden verschiedene Formeln zur Berechnung der zu erwartenden Zeitdauer vom Schließen einer Schleife bis zu deren Erkennung durch RIP-MTI entwickelt. Bei der Überprüfung der Formeln hat sich gezeigt, dass diese Berechnungen die Realität recht genau beschreiben. Das erste gesetzte Ziel wurde also erreicht.

Anschließend wurden die Probleme der Schleifenerkennung genauer untersucht. Mit Slow Start wurde eine Erweiterung entwickelt, um die Erkennung der Schleifen zu verbessern, so dass weniger Probleme auftreten. In der Startphase kann die Schleifenerkennung mit Slow Start wesentlich robuster erfolgen. Jedoch bleiben die erkannten Schleifen nicht dauerhaft gespeichert. Dieses Problem kann aber im Rahmen zukünftiger Forschungsarbeit behoben werden.

Momentan liegt die Stärke von Slow Start in dem wesentlich verbesserten Konvergenzverhalten des Netzwerkes nach dem Start eines neuen Routers.
Bisher scheint die Startphase eines Routers noch nicht genauer betrachtet worden zu sein. Anders ist nicht zu erklären, dass die direkt angeschlossenen Netzwerke wie in Kapitel 4.3.2 gezeigt unmittelbar nach dem Start des Routers nur über ein Interface per Triggered Update verbreitet werden. Auch eine recht hohe Belastung der Netzwerke durch den Routerstart wurde gebilligt.
Durch die Verwendung von Slow Start können hier mit einfachen Mitteln deutliche Verbesserungen erzielt werden. So lässt sich die Anzahl der nach dem Start eines Routers verschickten Update-Nachrichten bis zur Konvergenz des Netzwerkes etwa halbieren, wobei die Zeitdauer der Konvergenz gleich bleibt. Der in Kapitel 3.3.4 erwähnte Nachteil der längeren Konvergenzzeit ist folglich nicht vorhanden.
Es zeigt sich, dass auch das zweite Ziel erreicht wurde. Mit der Unterdrückung von Update-Nachrichten in Netzwerken, in denen sich kein weiterer Router befindet, bietet sich jedoch noch eine weitere Möglichkeit, um Slow Start im Rahmen zukünftiger Forschungsarbeit zu optimieren.

Literaturverzeichnis

BOH08 BOHDANOWICZ, FRANK: *Weiterentwicklung und Implementierung des RIP-MTI-Routing-Daemons*. Koblenz-Landau, Universität, Fachbereich 4, Dipl.-Arb., 2008

FEFE VON LEITNER, FELIX: *Multicast*. URL http://www.fefe.de/ct/multicast.txt. – Aktualisierungsdatum: 28.08.2001. – Dateigröße: 16315 Bytes.

FLO94 FLOYD, SALLY; JACOBSON, VAN: *The Synchronization of Periodic Routing Messages*, Lawrence Berkeley Laboratory, 1994

HUN03 HUNT, ANDREW; THOMAS, DAVID: *Der pragmatische Programmierer.* 1. Aufl. München: Hanser, 2003. – Deutsche Übersetzung, Original: The Pragmatic Programmer - From Jouneyman to Master. – ISBN 3-446-22309-6

KLE01 KLEEMANN, THOMAS: *RIPeval – Evaluierung und Weiterentwicklung des RIP-MTI-Algorithmus*. Koblenz-Landau, Universität, Fachbereich 4, Dipl.-Arb., 2001

LAN07 LANGE, STEFAN: *Zentrale Betrachtung von Routing-Informationen zur Analyse des Konvergenzverhaltens verschiedener RIP-Algorithmen und Unterstützung des Generierens von Testfällen*. Koblenz-Landau, Universität, Fachbereich 4, Dipl.-Arb., 2007

MON07 MONREAL, MICHAEL: *Simulation mit VNUML*. Koblenz-Landau, Universität, Fachbereich 4, Studienarbeit, 2007

RFC1058 HEDRICK, C.: *Routing Information Protocol*, 1988. – RFC 1058

RFC1518 REKHTER, Y.; LI, T.: *An Architecture for IP Address Allocation with CIDR*, 1993. – RFC 1518

RFC1519 FULLER, V.; LI, T.; YU, J.; VARADHAN, K.: *Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy*, 1993. – RFC 1519

RFC1826 ATKINSON, R.: *IP Authentication Header*, 1995. – RFC 1826

RFC2080 MALKIN, G.; MINNEAR, R.: *RIPng for IPv6*, 1997. – RFC 2080

RFC2453 MALKIN, G.: *RIP Version 2*, 1998. – RFC 2453

SCH99 SCHMID, ANDREAS: *RIP-MTI: Minimum-effort loop-free distance vector routing algorithm*. Koblenz-Landau, Universität, Fachbereich 4, Dipl.-Arb., 1999

SOT08 SOTIROV, ALEXANDER; STEVENS, MARC; APPELBAUM, JACOB; LENSTRA, ARJEN; MOLNAR, DAVID; OSVIK, DAG ARNE; DE WEGER, BENNE: *MD5 considered harmful today.* URL http://www.win.tue.nl/hashclash/rogue-ca/. – Aktualisierungsdatum: 26.01.2009. – Dateigröße: 140218 Bytes.

TAN04 TANENBAUM, ANDREW S.: *Computernetzwerke*. 4. überarbeitete Aufl. München: Pearson, 2004. – Deutsche Übersetzung, Original: Computer Networks. – ISBN 3-8273-7046-9

WIKI02 Seite „*Bellman-Ford-Algorithmus*“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 2. Januar 2009, 16:16 UTC. URL http://de.wikipedia.org/w/index.php?title=Bellman-Ford-Algorithmus&oldid=54819110 (Abgerufen: 13. Januar 2009, 16:47 UTC)

WIKI03 Seite „*Arpanet*“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 14. Januar 2009, 09:44 UTC. URL http://de.wikipedia.org/w/index.php?title=Arpanet&oldid=55310584 (Abgerufen: 23. Januar 2009, 12:15 UTC)

Abbildungsverzeichnis

Glossar

ARPANET
Advanced Research Projects Agency Network. Ein frühes amerikanisches Computernetzwerk, der Vorläufer des Internets.

CTI
Counting to infinity. Ein Konvergenzproblem des Distanz-Vektor-Routings.

Daemon
Unix-Bezeichnung für einen Dienst, also ein Programm, welches im Hintergrund läuft und als Server fungiert.

Distanz-Vektor-Routing
Eine Methode für den Austausch von Routinginformationen in Computernetzwerken. Bekannte Protokolle sind RIP und IGRP. Das Gegenteil zum Distanz-Vektor-Routing ist das Pfad-Vektor-Routing.

IGRP
Interior Gateway Routing Protocol, ein von Cisco Systems entwickeltes Routingprotokoll. Ziel von IGRP war eine bessere Skalierbarkeit als bei RIP. IGRP wird heute kaum noch eingesetzt. Sein Nachfolger ist EIGRP (Enhanced Interior Gateway Routing Protocol)

Konvergenz
Ein Netzwerk ist konvergent, wenn alle Router eine einheitliche Sicht auf das Netzwerk haben.

Latenzzeit
Verzögerungszeit. Hier die Zeitdauer, die ein Datenpaket benötigt, um über ein Netzwerk übertragen zu werden.

Quagga
Eine quelloffene Routing-Suite. Es werden die Routingprotokolle RIP (Routing Information Protocol), OSPF (Open Shortest Path First) und BGP (Border Gateway Protocol) unterstützt.

Request-Nachricht
Nachricht zum manuellen Abfragen der Routingtabelle (oder einzelner Einträge) eines Routers.

RIP
Routing Information Protocol. Ein Routingprotokoll, das auf dem Distanz-Vektor-Routing basiert. Es gibt verschiedene Versionen: RIPv1, RIPv2, RIPng sowie RIP-MTI.

RIP-MTI
Routing Information Protocol with Metric Based Topology Investigation. Erweiterung von RIP zur Verhinderung von CTIs, kompatibel zu RIP.

Routingschleife
Die Router innerhalb einer Routingschleife verweisen gegenseitig auf sich. Grundlage einer Routingschleife ist eine topologische Schleife.

SL_Client
Der SL_Client schickt dem SL_Server Informationen zum aktuellen Zustand des RIP-Daemons.

SL_Server
Der SL_Server stellt eine Schnittstelle bereit, mit der sich ein oder mehrere SL_Clients verbinden und Informationen über den aktuellen Status des RIP-Routers bereitstellen können.

Slow Start
Eine in dieser Arbeit vorgestellte Erweiterung für RIP und RIP-MTI, um das Startverhalten des Routers und die initiale Schleifenerkennung von RIP-MTI zu verbessern.

Timed Update
Regelmäßiger Austausch von Routinginformationen zwischen den Routern.

Topologische Schleife
Eine redundant ausgelegte hardwaremäßige Verbindung zwischen mehreren Routern.

Triggered Update
Nachrichten zum Austausch von Routinginformationen, falls die Routingtabelle geändert wurde.

UDP
User Datagram Protocol, ein verbindungsloses Netzwerkprotokoll. Dabei werden Nachrichten über das Netzwerk verschickt, ohne dass der Empfang der Nachrichten bestätigt wird.

UML
User Mode Linux. UML erlaubt die Ausführung des Linux Kernels im Benutzermodus, so dass auf einem Linux System mehrere virtuelle Maschinen betrieben werden können. Weiterhin steht UML für Unified Modeling Language. Hierbei handelt es sich um Diagramme zur Spezifikation von Software.

Update-Nachricht
Nachricht mit den Informationen der Routingtabelle. Update-Nachrichten werden entweder als Timed Update oder als Triggered Update verschickt.

VM
Virtuelle Maschine. Eine durch Software simulierte Laufzeitumgebung, so dass ein eigenes, vom Hostsystem und dessen Hardware unabhängiges Betriebssystem installiert werden kann.

VNUML
Virtual Network User Mode Linux. Ein Tool zur einfachen Generierung von Netzwerken aus mehreren per UML erzeugen VMs.

XML
Extensible Markup Language. Eine Sprache zur Darstellung von strukturierten Daten.

XT_Client
Der XT_Client verbindet sich mit dem XT_Server. Über diesen kann der Benutzer einen RIP-Router von außen steuern.

XT_Server
Der XT_Server stellt eine Schnittstelle zur Verfügung über welche sich der RIP-Daemon mit einem XT_Client steuern lässt.

XTPeer
Der XTPeer bietet eine graphische Benutzeroberfläche für den XT_Client sowie den SL_Server. Hierdurch können mit dem XTPeer verbundene Router gesteuert sowie deren Informationen angezeigt werden.

Anhang A Verzögerung bei Triggered Updates

A.1 Berechnung des Erwartungswertes

Als Wahrscheinlichkeitsdichtefunktion wurde in Kapitel 3.1.3 (Abbildung 3.2) folgende Funktion abgebildet:

$$f(x)=\begin{cases} 0 & \text{für } x<0 \\ -\frac{2}{25}x+\frac{2}{5} & \text{für } 0\le x\le 5 \\ 0 & \text{für } x>5 \end{cases} \tag{A.1}$$

Diese ergibt sich folgendermaßen: Wie in Kapitel 3.1.3 erläutert, liegt die Verzögerung zwischen 0 und 5 Sekunden. Entsprechend ist die Wahrscheinlichkeit für eine Verzögerung außerhalb dieses Intervalls null. Weiterhin wurde gezeigt, dass die Wahrscheinlichkeit für größere Verzögerungen streng monoton fällt. Da $\int_{-\infty}^{\infty} f(x)dx=1$ sein muss, kann Formel A.1 leicht berechnet werden.

Um den Erwartungswert zu berechnen, wird die Wahrscheinlichkeitsdichtefunktion in die Formel für den Erwartungswert (A.2, erste Zeile) eingesetzt:

$$\begin{aligned} E(X)=\mu &= \int_{-\infty}^{\infty} x\cdot f(x)\,dx \\ E(X)=\mu &= \int_{0}^{5} x\cdot\left(-\frac{2}{25}\cdot x+\frac{2}{5}\right)dx \\ &= \int_{0}^{5} -\frac{2}{25}\cdot x^2+\frac{2}{5}\cdot x\,dx \\ &= \left(-\frac{2}{3}\cdot 5^3+\frac{1}{5}\cdot 5^2\right)-\left(-\frac{2}{3}\cdot 0^3+\frac{1}{5}\cdot 0^2\right) \\ &= \frac{5}{3} \end{aligned} \tag{A.2}$$

A.2 Berechnung der Standardabweichung

Genauso wird bei der Berechnung der Varianz verfahren. Der in A.2 berechnete Erwartungswert und die Wahrscheinlichkeitsdichtefunktion werden in die allgemeine Formel (A.3, erste Zeile) eingesetzt:

$$\begin{aligned} Var(X) &= \int_{-\infty}^{\infty} (x-\mu)^2 \cdot f(x)\,dx \\ Var(X) &= \int_{0}^{5} \left(x-\frac{5}{3}\right)^2 \cdot \left(-\frac{2}{25}\cdot x+\frac{2}{5}\right) dx \\ &= \int_{0}^{5} -\frac{2}{25}\cdot x^3+\frac{2}{3}\cdot x^2-\frac{14}{9}\cdot x+\frac{10}{9}\,dx \\ &= -\frac{1}{50}\cdot 5^4+\frac{2}{9}\cdot 5^3-\frac{7}{9}\cdot 5^2+\frac{10}{9}\cdot 5 \\ &= \frac{25}{18} \end{aligned} \tag{A.3}$$

Aus der Varianz kann nun die Standardabweichung berechnet werden:

$$\begin{aligned} \sigma &= \sqrt{Var(X)} \\ \sigma &= \sqrt{\frac{25}{18}} = \frac{5}{3\sqrt{2}} \end{aligned} \tag{A.4}$$

Anhang B Verzögerung bei Timed Updates

B.1 Berechnung des Erwartungswertes

Die Funktionsgleichung der Wahrscheinlichkeitsdichtefunktion aus Abbildung 3.3 lautet:

$$f(x)=\begin{cases} 0 & \text{für } x<0 \\ \frac{1}{30} & \text{für } 0\le x\le 25 \\ -\frac{1}{300}\cdot x+\frac{7}{60} & \text{für } 25<x\le 35 \\ 0 & \text{für } x>35 \end{cases} \tag{B.1}$$

In Kapitel 3.1.3 wurde gezeigt, dass die Verzögerung zwischen 0 und 35 Sekunden liegen muss. Deshalb ist die Wahrscheinlichkeit für eine Verzögerung außerhalb dieses Intervalls null. Verzögerungen von 0 bis 25 Sekunden sind gleichwahrscheinlich, danach fällt die Wahrscheinlichkeit streng monoton. Da $\int_{-\infty}^{\infty} f(x)dx=1$ sein muss, kann Formel B.1 leicht berechnet werden.

Folgend die Berechnung des Erwartungswertes. In der ersten Zeile steht die allgemeine Formel. Dort muss die Wahrscheinlichkeitsdichtefunktion eingesetzt werden:

$$\begin{aligned} E(X)=\mu &= \int_{-\infty}^{\infty} x\cdot f(x)\,dx \\ E(X)=\mu &= \int_{0}^{25} x\cdot\left(\frac{1}{30}\right)dx+\lim_{t\to 25+}\int_{t}^{35} x\cdot\left(-\frac{1}{300}\cdot x+\frac{7}{60}\right)dx \\ &= \int_{0}^{25}\frac{1}{30}\cdot x\,dx+\lim_{t\to 25+}\int_{t}^{35}-\frac{1}{300}\cdot x^2+\frac{7}{60}\cdot x\,dx \\ &= \frac{1}{60}\cdot 25^2+\left(-\frac{1}{900}\cdot 35^3+\frac{7}{120}\cdot 35^2\right)-\lim_{t\to 25+}\left(-\frac{1}{900}\cdot t^3+\frac{7}{120}\cdot t^2\right) \\ &= \frac{545}{36} \\ &= 15\frac{5}{36} \end{aligned} \tag{B.2}$$

B.2 Berechnung der Standardabweichung

Um die Varianz zu berechnen, werden in die allgemeine Formel (B.3, erste Zeile) der in B.2 berechnete Erwartungswert sowie die Wahrscheinlichkeitsdichtefunktion aus B.1 eingesetzt.

$$\begin{aligned} Var(X) &= \int_{-\infty}^{\infty} (x-\mu)^2 \cdot f(x)\,dx \\ Var(X) &= \int_{0}^{25} \left(x-\frac{545}{36}\right)^2 \cdot \frac{1}{30}\,dx + \lim_{t\to 25+} \int_{t}^{35} \left(x-\frac{545}{36}\right)^2 \cdot \left(-\frac{1}{300}\cdot x+\frac{7}{60}\right) dx \\ &= \frac{307725}{3888} \end{aligned} \tag{B.3}$$

Aus der Varianz lässt sich die Standardabweichung berechnen:

$$\begin{aligned} \sigma &= \sqrt{Var(X)} \\ \sigma &= \sqrt{\frac{307725}{3888}} = \frac{5}{36}\cdot\sqrt{4103} \end{aligned} \tag{B.4}$$

Anhang C Erwartungswert und Standardabweichung bei der stetigen Gleichverteilung

Folgend die Formeln zur Berechnung des Erwartungswertes sowie der Standardabweichung für die stetige Gleichverteilung (auch Rechtecksverteilung genannt).

C.1 Formel für den Erwartungswert

Allgemeine Formel für den Erwartungswert:

$$E(X)=\mu=\int_{-\infty}^{\infty} x\,f(x)dx \tag{C.1}$$

Bei der stetigen Gleichverteilung auf dem Intervall $[a,b]$ ist als Wahrscheinlichkeitsdichtefunktion folgende Funktion gegeben:

$$f(x)=\begin{cases} 0 & x\le a \\ \dfrac{1}{b-a} & a<x<b \\ 0 & x\ge b \end{cases} \tag{C.2}$$

Wird Formel C.2 in Formel C.1 eingesetzt und das Integral mittels der Verteilungsfunktion (C.3) berechnet, ergibt sich Formel C.4 zur einfachen Berechnung des Erwartungswertes bei stetigen gleichverteilten Zufallsgrößen.

$$E(x)=\begin{cases} 0 & x\le a \\ \dfrac{x-a}{b-a} & a<x<b \\ 1 & x\ge b \end{cases} \tag{C.3}$$

$$E(X)=\mu=\frac{a+b}{2} \tag{C.4}$$

C.2 Formel für die Standardabweichung

Die Standardabweichung ergibt sich als Wurzel der Varianz, wobei sich die Varianz allgemein folgendermaßen berechnet:

$$Var(X) = E(X^2) - (E(X))^2 \tag{C.5}$$

Wird die Varianz nun für die Wahrscheinlichkeitsdichtefunktion aus Formel C.2 berechnet, so ergibt sich vereinfacht:

$$Var(X) = \frac{(b-a)^2}{12} \tag{C.6}$$

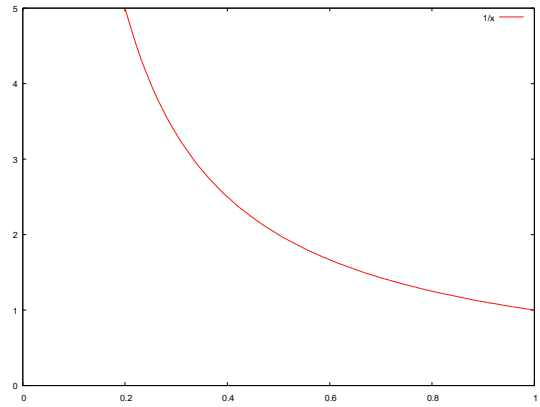
Hieraus muss die Wurzel gezogen werden, um die Standardabweichung zu erhalten:

$$\sigma = \sqrt{\frac{(b-a)^2}{12}} = \frac{b-a}{2\sqrt{3}} \tag{C.7}$$

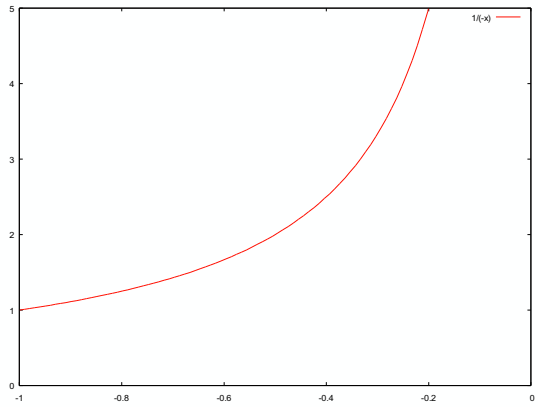
Anhang D Funktionsherleitung: Erwartungswert der Verzögerung bei Paketverlusten

Dieser Anhang zeigt die Herleitung der Formel 3.3 von Seite 27 zur Berechnung der zu erwartenden Verzögerung bei verlustbehafteter Übertragung von Update-Nachrichten.

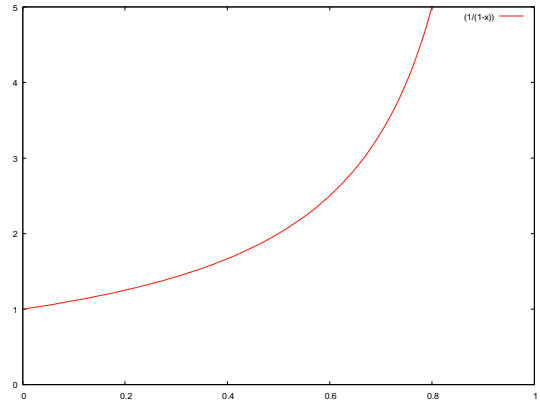
In Abbildung D.1 auf der folgenden Seite ist die Entwicklung der Formel skizziert. Da die gesuchte Formel für $x \to 1,\ x < 1$ gegen unendlich gehen soll, liegt es nahe mit der Kehrwertfunktion $f(x) = 1/x$ zu beginnen (Abbildung D.1.a). Diese Funktion geht für $x \to 0,\ x > 0$ gegen unendlich. Deshalb wird sie zuerst an der y-Achse gespiegelt (Abbildung D.1.b). Zusammen mit einer Verschiebung um 1 nach rechts (Abbildung D.1.c) geht die Funktion für $x \to 1,\ x < 1$ bereits gegen unendlich. Noch beginnt die Funktion nicht im Ursprung, daher muss sie um 1 nach unten verschoben werden (Abbildung D.1.d). Der Funktionswert $f(0{,}5) = 1$ lässt annehmen, dass eine Streckung um den Faktor 30 die gesuchte Funktion ergibt (Abbildung D.1.e). Eine kurze Probe mit den Werten aus Tabelle 3.1 zeigt, dass es sich tatsächlich um eben jene Funktion handelt. Nun muss nur noch die Variable x durch p_v, die Wahrscheinlichkeit für den Verlust von Nachrichten im Netzwerk, ersetzt werden. Anschließend kann nach Formel 3.3 für jede Verlustwahrscheinlichkeit von Update-Nachrichten die zu erwartende Verzögerung berechnet werden.



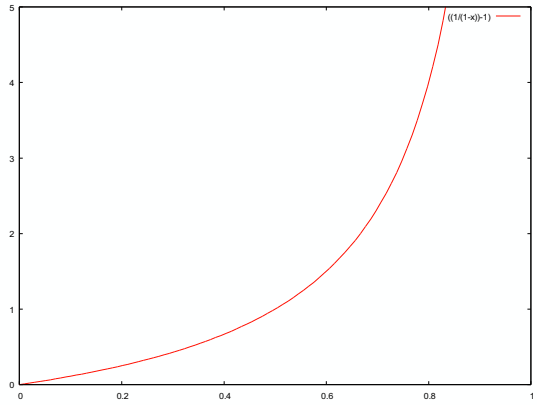
a) Ausgangsfunktion: f(x) = 1/x



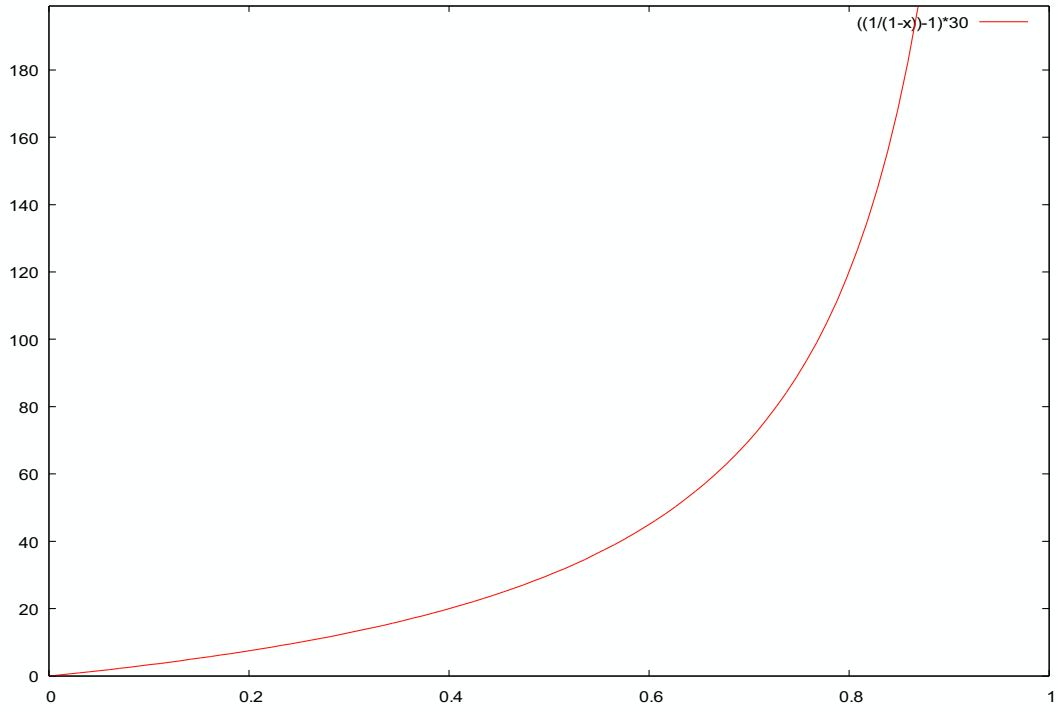
b) Spiegelung an der y-Achse: f(x) = 1/-x



c) Verschiebung um 1 nach rechts: f(x) = 1/(1-x)



d) Verschiebung um 1 nach unten: f(x) = (1/(1-x))-1



*e) Streckung um den Faktor 30: f(x) = ((1/(1-x))-1)*30*

Abbildung D.1: Entwicklung der Funktion zur Berechnung der zu erwartenden Verzögerung.

Anhang E Quellcode und Beschreibung von Slow Start

Folgend nun die bei der Implementierung von Slow Start vorgenommenen Änderungen im Quellcode des RIP-MTI (Anhang E.1 bis Anhang E.4) sowie des RIP (Anhang E.5). Die grau unterlegten Zeilennummern beziehen sich jeweils auf die Zeilennummern des Originalquelltextes von RIP-MTI bzw. RIP, wie er auf der CD-ROM im Anhang zu finden ist.

E.1 Quellcode und Beschreibung der Datei ripd.h

Zunächst werden die Definitionen in der Datei `ripd.h` beschrieben, wie sie in Quellcode E.1 aufgeführt sind. In den Zeilen `98` bis `108` werden erst einmal verschiedene Konstanten definiert, damit spätere Überprüfungen und Zuweisungen übersichtlicher wirken. Hier nun die Bedeutungen der Konstanten:

- **RIP_SLOW_START_IF_SILENT:** Das Interface ist stumm geschaltet. Es können dort keinerlei Nachrichten gesendet werden.
- **RIP_SLOW_START_IF_REQUEST:** Über das entsprechende Interface können nur Request-Nachrichten verschickt werden.
- **RIP_SLOW_START_IF_UPDATE:** Das Interface befindet sich im normalen Betriebsmodus. Es können Update- und Request-Nachrichten verschickt werden.
- **RIP_SLOW_START_DISABLED:** Slow Start ist inaktiv.
- **RIP_SLOW_START_ENABLED:** Slow Start ist aktiv.
- **RIP_SLOW_START_START:** Der Router befindet sich in der Startphase (Sammeln von Informationen, kein Versand von Nachrichten).
- **RIP_SLOW_START_RUNNING:** Der Router befindet sich in der normalen Betriebsphase.
- **RIP_SLOW_START_UPDATES:** Diese Konstante wurde zu Testzwecken eingeführt. Sie speichert die Anzahl der Update-Intervalle, bis der Router bei aktiviertem Slow Start von der Startphase in die Betriebsphase übergeht. Sie sollte deshalb normalerweise auf den Wert 1 gesetzt bleiben, so dass der Router lediglich im ersten Updateintervall nach dem Einschalten keine Update-Nachrichten verschickt. Ein Wert größer 1 verlängert die Startphase unnötig und ist deshalb nur zu Testzwecken sinnvoll. Wird diese Konstante auf einen kleineren Wert als 1 gesetzt, so geht der Router direkt in die Betriebsphase über. Slow Start ist somit dauerhaft deaktiviert.

Anschließend werden in den Zeilen 117 bis 119 dem Konstrukt rip drei Attribute beigefügt. Dieses Konstrukt ist im späteren Programmablauf global definiert, so dass es von überall erreichbar ist.

- **slow_start_state:** Wurde Slow Start aktiviert, so wird dieses Attribut auf den Wert RIP_SLOW_START_ENABLED gesetzt. Anderenfalls nimmt es den Wert RIP_SLOW_START_DISABLED an.
- **slow_start_mode:** Dieses Attribut enthält einen von zwei möglichen Werten, welche den Zustand von Slow Start beschreiben: entweder RIP_SLOW_START_START (Router in der Startphase) oder RIP_SLOW_START_RUNNING (Router in der Betriebsphase).
- **slow_start_updates:** In diesem Attribut wird die aktuelle Anzahl der Update-Intervalle seit dem Start des Routers gespeichert.

```
...
 97  /* RIP SlowStart */
 98  #define RIP_SLOW_START_IF_SILENT   0
 99  #define RIP_SLOW_START_IF_REQUEST  1
100  #define RIP_SLOW_START_IF_UPDATE   2
101
102  #define RIP_SLOW_START_DISABLED    0
103  #define RIP_SLOW_START_ENABLED     1
104
105  #define RIP_SLOW_START_START       0
106  #define RIP_SLOW_START_RUNNING     1
107
108  #define RIP_SLOW_START_UPDATES     1
109
110  /* RIP structure. */
111  struct rip
112  {
...
116    /* SlowStart */
117    int slow_start_state;
118    int slow_start_mode;
119    int slow_start_updates;
...
175  };
...
307  /* RIP specific interface configuration. */
308  struct rip_interface
309  {
310    /* SlowStart */
311    int slow_start_mode;
...
364  };
...
```

Quellcode E.1: ripd.h

Zuletzt hat noch in Zeile 311 das Konstrukt rip_interface eine zusätzliches Attribut bekommen. Es heißt ebenfalls slow_start_mode und nimmt einen der folgenden drei Werte an: RIP_SLOW_START_IF_SILENT (Interface darf keine Nachrichten verschicken), RIP_SLOW_START_IF_REQUEST (Interface darf nur Request-Nachrichten verschicken) oder RIP_SLOW_START_IF_UPDATE (Interface darf alle Nachrichten verschicken).
Dieses Attribut wurde zu Testzwecken eingeführt und wird in der aktuellen Version nicht mehr ausgewertet.

E.2 Quellcode und Beschreibung der Datei rip_interface.c

In der Datei rip_interface.c (Quellcode E.2) wurden in der Funktion rip_interface_new, welche ein neues Interface initialisiert, die Zeilen 138 und 139 hinzugefügt. Und zwar wird hier das in ripd.h (Quellcode E.1) definierte Attribut slow_start_mode aus dem Konstrukt rip_interface auf den initalen Wert RIP_SLOW_START_IF_REQUEST gesetzt, so dass auf dem neuen Interface zu Beginn nur Request-Nachrichten verschickt werden können.
Weiterhin wird eine Meldung über den Status des Interfaces in die Logdatei aufgenommen.

```
...
116  /* Allocate new RIP's interface configuration. */
117  static struct rip_interface *
118  rip_interface_new (void)
119  {
120    struct rip_interface *ri;
...
137    /* SlowStart */
138    ri->slow_start_mode = RIP_SLOW_START_IF_REQUEST;
139    zlog_debug ("SlowStart new Interface in RequestMode!");
140
141    return ri;
142  }
...
```

Quellcode E.2: rip_interface.c

E.3 Quellcode und Beschreibung der Datei ripd.c

Die meisten Änderungen ergeben sich in der Datei ripd.c. Sie sind in Quellcode E.3 dargestellt.

In der Funktion rip_read werden ankommende Nachrichten (z.B. Requests und Updates) verarbeitet. Erhält der Router auf einem Interface eine Update-Nachricht, so wird das entsprechende Interface auf den Status RIP_SLOW_START_IF_SILENT gesetzt

(Zeile 2286).
Erhält der Router hingegen eine Request-Nachricht, so wird das Interface in den Status RIP_SLOW_START_IF_UPDATE versetzt, sofern es sich zuvor im Status RIP_SLOW_START_IF_REQUEST befand (Zeilen 2294 und 2295).

In den Zeilen 2878 bis 2897 wurde eine neue Funktion mit dem Namen slowstart_modechange programmiert. Diese wird im späteren Programmverlauf über einen Timer aufgerufen und prüft, ob sich der Router noch in der Startphase befindet. Innerhalb der Funktion wird zuerst in Zeile 2882 der Timer gelöscht und eine Meldung über den Aufruf der Funktion in die Logdatei geschrieben (Zeile 2883). In den Zeilen 2886 und 2887 wird die Variable slow_start_updates des Konstrukts rip inkrementiert. In dieser Variablen wird – wie bereits zu Quellcode E.1 beschrieben – die Anzahl der Updateintervalle seit dem Einschalten des Routers gezählt. Damit es hier nicht zu Problemen mit dem Übertrag kommt, wenn der Router lange läuft, findet zuerst eine Überprüfung statt, ob der Wert überhaupt noch weiter inkrementiert werden muss. Eine Zählung der Intervalle erfolgt nur, wenn sich der Router noch in der Startphase befindet.
Anschließend wird in den Zeilen 2890 bis 2896 überprüft, ob sich der Router noch in der Startphase befindet und ob sie in diesem Moment beendet wird. Ist letzteres der Fall, wird in Zeile 2294 das Attribut slow_start_mode auf den Wert RIP_SLOW_START_RUNNING gesetzt.

In der Funktion rip_triggered_update, welche aufgerufen wird, wenn ein Triggered Update verschickt werden soll, muss nun geprüft werden, ob Slow Start aktiv ist und sich in der Startphase befindet. Für diesen Fall wird eine Debug-Meldung ausgegeben und die Funktion beendet (Zeilen 2992 bis 2996).

In den Zeilen 3058 bis 3097 wird das Konstrukt rip in der Funktion rip_create initialisiert. Hier müssen die in ripd.h hinzugefügten Attribute mit Default-Werten versehen werden. In Zeile 3074 wird Slow Start standardmäßig ausgeschaltet. Dies ist notwendig, damit Slow Start inaktiv ist, wenn der Router ohne den Startparameter -s gestartet wird. Die folgende Zeile versetzt den Router in die Startphase. In Zeile 3076 wird der Zähler für die Updateintervalle seit dem Start mit 0 initialisiert.

Abschließend muss noch die Funktion rip_event geändert werden. Hier muss die schon zuvor beschriebene Funktion slowstart_modechange aufgerufen werden. Dies geschieht immer, bevor ein Timed Update verschickt wird. Entsprechend wird in Zeile 3259 der Timer zum Aufruf dieser Funktion auf den Wert des Update-Intervalls gesetzt.

```
 ...
2020  /* First entry point of RIP packet. */
2021  static int
2022  rip_read (struct thread *t)
2023  {
 ...
2028    struct rip_packet *packet;
 ...
2034    struct rip_interface *ri;
 ...
2280    /* Process each command. */
2281    switch (packet->command)
2282    {
2283      case RIP_RESPONSE:
2284
2285        /* SlowStart response received. */
2286        ri->slow_start_mode = RIP_SLOW_START_IF_SILENT;
 ...
2289        break;
2290      case RIP_REQUEST:
2291      case RIP_POLL:
2292
2293        /* SlowStart request received. */
2294        if(ri->slow_start_mode == RIP_SLOW_START_IF_REQUEST)
2295          ri->slow_start_mode = RIP_SLOW_START_IF_UPDATE;
 ...
2298        break;
 ...
2314    }
 ...
2317  }
 ...
2878  /* SlowStart Mode Change */
2879  void
2880  slowstart_modechange (void)
2881  {
2882    rip->t_slow_start_modechange = NULL;
2883    zlog_debug("SlowStart modechange");
2884
2885    /* SlowStart, count updates */
2886    if(rip->slow_start_updates <= RIP_SLOW_START_UPDATES)
2887      rip->slow_start_updates++;
2888
2889    /* SlowStart is now changing from StartMode to RunningMode. */
2890    if(rip->slow_start_updates >= RIP_SLOW_START_UPDATES &&
2891        rip->slow_start_state == RIP_SLOW_START_ENABLED &&
2892        rip->slow_start_mode != RIP_SLOW_START_RUNNING) {
2893      zlog_debug("SlowStart now in running mode");
2894      rip->slow_start_mode = RIP_SLOW_START_RUNNING;
2895    } else
2896      rip_event (RIP_SLOWSTART_MODECHANGE, 0);
2897  }
 ...
2975  /* Execute triggered update. */
2976  static int
2977  rip_triggered_update (struct thread *t)
2978  {
 ...
```

```
2992      /* If SlowStart is enabled and we are in Start-Mode no triggerd Update should be sent. */
2993      if(rip->slow_start_state == RIP_SLOW_START_ENABLED &&
      rip->slow_start_mode == RIP_SLOW_START_START) {
2994          zlog_debug ("SlowStart triggered update canceled!");
2995          return 0;
2996      }
 ...
3020    }
 ...
3058    /* Create new RIP instance and set it to global variable. */
3059    static int
3060    rip_create (void)
3061    {
3062      rip = XMALLOC (MTYPE_RIP, sizeof (struct rip));
3063      memset (rip, 0, sizeof (struct rip));
 ...
3073      /* SlowStart */
3074      rip->slow_start_state = RIP_SLOW_START_DISABLED;
3075      rip->slow_start_mode = RIP_SLOW_START_START;
3076      rip->slow_start_updates = 0;
 ...
3097    }
 ...
3238    void
3239    rip_event (enum rip_event event, int sock)
3240    {
3241      int jitter = 0;
 ...
3244      switch (event)
3245      {
 ...
3249        case RIP_UPDATE_EVENT:
3250          jitter = rip_update_jitter (rip->update_time);
3251
3252          /* SlowStart Modechange */
3253          if(rip->t_slow_start_modechange) {
3254            thread_cancel (rip->t_slow_start_modechange);
3255            rip->t_slow_start_modechange = NULL;
3256          }
3257          if(rip->slow_start_updates < RIP_SLOW_START_UPDATES)
3258            rip->t_slow_start_modechange =
3259                thread_add_timer (master, slowstart_modechange,
      NULL, rip->update_time + jitter - 1);
 ...
3270          break;
 ...
3292      }
3293    }
 ...
```

Quellcode E.3: ripd.c

E.4 Quellcode und Beschreibung der Datei rip_main.c

Zuerst werden in Zeile `55` von Quellcode E.4 im Konstrukt `longoptions` die Option `slow` sowie der Shortcut „s“ definiert. Weiterhin wird die Funktion `usage` in Zeile

123 angepasst, so dass die Slow Start Funktion bei einem Aufruf der Programmhilfe angezeigt wird.

Anschließend müssen noch verschiedene Änderungen in der main-Funktion vorgenommen werden:

- In Zeile 207 wird die Variable slow_start deklariert und auf den Wert RIP_SLOW_START_DISABLED gesetzt. So lange also RIP nicht mit der Option -s bzw. --slow gestartet wird, ist Slow Start inaktiv.
- Zeile 229 muss angepasst werden, damit der Parameter --slow beim Programmaufruf geparst und keine Fehlermeldung ausgegeben wird.
- Der Switch-Anweisung aus Zeile 234 muss ein weiterer Fall (Zeilen 270 bis 273) hinzugefügt werden. Dieser Fall wird betreten, wenn RIP mit der Option -s bzw. --slow gestartet wird. Slow Start muss also aktiviert werden. Hierzu wird einfach die Variable slow_start auf den Wert RIP_SLOW_START_ENABLED gesetzt.
- In Zeile 339, nachdem das Konstrukt rip global initialisiert wurde, wird gespeichert, ob Slow Start nun aktiviert ist oder nicht. Hierzu wird einfach der zuvor gesetzte Wert der Variablen slow_start in das Attribut slow_start_state des Konstrukts rip geschrieben. In den folgenden beiden Zeilen wird eine Meldung in die Logdatei geschrieben, falls Slow Start aktiviert wurde.

```
...
 41  /* ripd options. */
 42  static struct option longopts[] =
 43  {
...
 55    { "slow", no_argument, NULL, 's'},
 56    { 0 }
 57  };
...
103  /* Help information display. */
104  static void
105  usage (char *progname, int status)
106  {
107    if (status != 0)
108      fprintf (stderr, "Try `%s --help' for more information.\n",
     progname);
109    else
110    {
111      printf ("Usage : %s [OPTION...]\n\
...
123        -s, --slow         Enable SlowStart\n\
...
127        Report bugs to %s\n", progname, ZEBRA_BUG_ADDRESS);
128    }
...
132    exit (status);
133  }
```

```
...
199  /* Main routine of ripd. */
200  int
201  main (int argc, char **argv)
202  {
...
207    int slow_start = RIP_SLOW_START_DISABLED;
...
224    /* Command line option parse. */
225    while (1)
226    {
227      int opt;
228
229      opt = getopt_long (argc, argv, "df:i:hA:P:u:g:rvC:s",
     longopts, 0);
230
231      if (opt == EOF)
232        break;
233
234      switch (opt)
235      {
...
270        case 's':
271          /* SlowStart */
272          slow_start = RIP_SLOW_START_ENABLED;
273          break;
...
291      }
292    }
...
338    /* SlowStart */
339    rip->slow_start_state = slow_start;
340    if(rip->slow_start_state == RIP_SLOW_START_ENABLED)
341      zlog_debug ("SlowStart enabled!");
...
349  }
...
```

Quellcode E.4: rip_main.c

E.5 Quellcode und Beschreibung: ripd.c der RIP-Implementierung

Folgend nun die Änderungen an RIP durch die Implementierung von Slow Start. Dargestellt werden nur die Unterschiede im Vergleich zur Implementierung von Slow Start in RIP-MTI. Als RIP-Daemon wurde die quelloffene Routingsoftware Quagga in der Version 0.99.11 verwendet.

Die in Quellcode E.3 gezeigte Funktion `slowstart_modechange` (Zeile `2880`) ist bei der Implementierung in RIP nicht notwendig. In dieser RIP-Version wird die Funktion `rip_update` nur aufgerufen, wenn ein Timed Update verschickt werden soll. Entsprechend kann der Statuswechsel dort vorgenommen werden (siehe Quellcode E.5). In den Zeilen `2584` und `2585` wird gezählt, wie oft diese Funktion aufgerufen wurde. In

den Zeilen 2588 bis 2590 wird geprüft, ob Slow Start beim Start des Routers aktiviert wurde und ob es sich noch in der Startphase befindet. Schließlich wird noch geschaut, ob die im Konstrukt rip vorgegebene Anzahl an Updateintervallen als Dauer der Startphase bereits erreicht wurde. Ist dies der Fall, wird in Zeile 2592 der Status des Routers auf die Betriebsphase umgestellt und eine entsprechende Meldung in die Logdatei geschrieben.

```
 ...
2576  /* RIP's periodical timer. */
2577  static int
2578  rip_update (struct thread *t)
2579  {
 ...
2583    /* SlowStart, count updates */
2584    if(rip->slow_start_updates <= RIP_SLOW_START_UPDATES)
2585      rip->slow_start_updates++;
2586
2587    /* SlowStart is now changing from StartMode to RunningMode. */
2588    if(rip->slow_start_updates == RIP_SLOW_START_UPDATES &&
2589        rip->slow_start_state == RIP_SLOW_START_ENABLED &&
2590        rip->slow_start_mode != RIP_SLOW_START_RUNNING) {
2591      zlog_debug("SlowStart now in running mode");
2592      rip->slow_start_mode = RIP_SLOW_START_RUNNING;
2593    }
 ...
2614  }
 ...
```

Quellcode E.5: ripd.c

Anhang F Validierung

In diesem Anhang sind die einzelnen XML-Dateien der eingesetzten VNUML-Simulationen abgebildet. Da die Dateien sehr ähnlich sind, ist nur die erste Simulation komplett dargestellt. Bei den folgenden Simulationen sind jeweils nur die Unterschiede zur der vollständigen Abbildung dargestellt.
Weiterhin sind die Ergebnisse der durchgeführten Tests zur Zeitdauer der Schleifenerkennung und der Effektivität von Slow Start aufgelistet.

F.1 VNUML-Simulationen

Folgend der gesamte Quellcode für die VNUML-Simulation in Kapitel 4.2.1 zur Validierung der Formeln für die zu erwartende Zeitdauer bis zur Erkennung einer Schleife in RIP-MTI. Mit der unter Quellcode F.1 dargestellten Simulation wird die Formel zur Berechnung des Erwartungswertes bei Szenario 1 validiert.

In den Zeilen 4 bis 17 werden die globalen Definitionen vorgenommen, welche für alle VMs gelten. Anschließend werden in den Zeilen 18 bis 23 die verschiedenen Netzwerke definiert. Ab Zeile 24 erfolgen die Definitionen der einzelnen VMs.

```
 1  <?xml version="1.0" encoding="UTF-8"?>
 2  <!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
 3  <vnuml>
 4    <global>
 5      <version>1.8</version>
 6      <simulation_name>TestSimulation</simulation_name>
 7      <automac />
 8      <vm_mgmt type="private" network="192.168.0.0" mask="24"
   offset="100">
 9        <host_mapping />
10      </vm_mgmt>
11      <vm_defaults exec_mode="mconsole">
12        <filesystem type="cow">/usr/local/share/vnuml/
   filesystems/miniuml</filesystem>
13        <kernel>/url/local/share/vnuml/kernels/linux</kernel>
14        <console id="0">xterm</console>
15        <forwarding />
16      </vm_defaults>
17    </global>
18    <net name="a" mode="virtual_bridge" type="lan" />
19    <net name="b" mode="virtual_bridge" type="lan" />
20    <net name="c" mode="virtual_bridge" type="lan" />
21    <net name="d" mode="virtual_bridge" type="lan" />
22    <net name="e" mode="virtual_bridge" type="lan" />
23    <net name="f" mode="virtual_bridge" type="lan" />
24    <vm name="R1">
25      <if id="1" net="a">
26        <ipv4>10.0.1.1</ipv4>
27      </if>
```

```
28      <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
29      <exec seq="start" type="verbatim">ripd -f
    /etc/quagga/zebra.conf -x 5000 -d</exec>
30      <exec seq="stop" type="verbatim">killall zebra</exec>
31      <exec seq="stop" type="verbatim">killall ripd</exec>
32    </vm>
33    <vm name="R2">
34      <if id="1" net="a">
35        <ipv4>10.0.1.2</ipv4>
36      </if>
37      <if id="2" net="b">
38        <ipv4>10.0.2.1</ipv4>
39      </if>
40      <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
41      <exec seq="start" type="verbatim">ripd -f
    /etc/quagga/zebra.conf -x 5000 -d</exec>
42      <exec seq="stop" type="verbatim">killall zebra</exec>
43      <exec seq="stop" type="verbatim">killall ripd</exec>
44    </vm>
45    <vm name="R3">
46      <if id="1" net="b">
47        <ipv4>10.0.2.2</ipv4>
48      </if>
49      <if id="2" net="c">
50        <ipv4>10.0.3.1</ipv4>
51      </if>
52      <if id="3" net="f">
53        <ipv4>10.0.6.1</ipv4>
54      </if>
55      <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
56      <exec seq="start" type="verbatim">ripd -f
    /etc/quagga/zebra.conf -x 5000 -d</exec>
57      <exec seq="stop" type="verbatim">killall zebra</exec>
58      <exec seq="stop" type="verbatim">killall ripd</exec>
59    </vm>
60    <vm name="R4">
61      <if id="1" net="c">
62        <ipv4>10.0.3.2</ipv4>
63      </if>
64      <if id="2" net="d">
65        <ipv4>10.0.4.1</ipv4>
66      </if>
67      <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
68      <exec seq="start" type="verbatim">ripd -f
    /etc/quagga/zebra.conf -x 5000 -d</exec>
69      <exec seq="stop" type="verbatim">killall zebra</exec>
70      <exec seq="stop" type="verbatim">killall ripd</exec>
71    </vm>
72    <vm name="R5">
73      <if id="1" net="e">
74        <ipv4>10.0.5.1</ipv4>
75      </if>
76      <if id="2" net="f">
77        <ipv4>10.0.6.2</ipv4>
78      </if>
```

```
 79      <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
 80      <exec seq="start" type="verbatim">ripd -f
    /etc/quagga/zebra.conf -x 5000 -d</exec>
 81      <exec seq="stop" type="verbatim">killall zebra</exec>
 82      <exec seq="stop" type="verbatim">killall ripd</exec>
 83    </vm>
 84  </vnuml>
```

Quellcode F.1: XML-Datei für VNUML zum Testen der Zeitdauer der Informationsübertragung bei Szenario 1.

In Quellcode F.2 ist ausschnittsweise die Simulation zur Validierung der zu erwartenden Zeitdauer bis zur Schleifenerkennung bei Szenario 2 dargestellt.

```
...
 21    <net name="d" mode="virtual_bridge" type="lan" />
 22    <vm name="R1">
...
 43    <vm name="R3">
 44      <if id="1" net="b">
 45        <ipv4>10.0.2.2</ipv4>
 46      </if>
 47      <if id="2" net="c">
 48        <ipv4>10.0.3.1</ipv4>
 49      </if>
 50      <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
...
 54    </vm>
 55    <vm name="R4">
...
 66    </vm>
 67  </vnuml>
```

Quellcode F.2: XML-Datei für VNUML zum Testen der Zeitdauer der Informationsübertragung bei Szenario 2.

Als erste Testsimulation für Slow Start wird die in Quellcode F.3 gezeigte VNUML-Simulation genutzt. Es sind wieder nur die Unterschiede zu Quellcode F.1 dargestellt.

```
...
 24    <vm name="R1">
...
 28      <if id="2" net="b">
 29        <ipv4>10.0.2.1</ipv4>
 30      </if>
...
 35    </vm>
 36    <vm name="R2">
...
 40      <if id="2" net="c">
 41        <ipv4>10.0.3.1</ipv4>
 42      </if>
 43      <if id="3" net="d">
 44        <ipv4>10.0.4.1</ipv4>
 45      </if>
...
 50    </vm>
```

```
 51     <vm name="R3">
...
 55       <if id="2" net="c">
 56         <ipv4>10.0.3.2</ipv4>
 57       </if>
 58       <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
...
 62     </vm>
 63     <vm name="R4">
 64       <if id="1" net="f">
 65         <ipv4>10.0.6.1</ipv4>
 66       </if>
 67       <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
...
 71     </vm>
...
 84     <vm name="R6">
 85       <if id="1" net="d">
 86         <ipv4>10.0.4.2</ipv4>
 87       </if>
 88       <if id="2" net="e">
 89         <ipv4>10.0.5.2</ipv4>
 90       </if>
 91       <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
 92       <exec seq="start2" type="verbatim">ripd -f
    /etc/quagga/zebra.conf -x 5000 -d</exec>
 93       <exec seq="stop" type="verbatim">killall zebra</exec>
 94       <exec seq="stop2" type="verbatim">killall ripd</exec>
 95     </vm>
 96   </vnuml>
```

Quellcode F.3: XML-Datei für VNUML. Testsimulation 1 für Slow Start.

Zuletzt in Quellcode F.4 ausschnittsweise die zweite Testsimulation für Slow Start.

```
...
 23     <net name="f" mode="virtual_bridge" type="lan" />
 24     <net name="g" mode="virtual_bridge" type="lan" />
 25     <net name="h" mode="virtual_bridge" type="lan" />
 26     <vm name="R1">
...
 30       <if id="2" net="h">
 31         <ipv4>10.0.8.1</ipv4>
 32       </if>
...
 37     </vm>
...
 50     <vm name="R3">
...
 54       <if id="2" net="c">
 55         <ipv4>10.0.3.1</ipv4>
 56       </if>
 57       <exec seq="start" type="verbatim">zebra -f
    /etc/quagga/zebra.conf -d</exec>
...
 61     </vm>
```

```
 62    <vm name="R4">
...
 69      <if id="3" net="g">
 70        <ipv4>10.0.7.1</ipv4>
 71      </if>
...
 76    </vm>
 77    <vm name="R5">
 78      <if id="1" net="d">
 79        <ipv4>10.0.4.2</ipv4>
 80      </if>
 81      <if id="2" net="e">
 82        <ipv4>10.0.5.1</ipv4>
 83      </if>
 84      <if id="3" net="f">
 85        <ipv4>10.0.6.1</ipv4>
 86      </if>
...
 91    </vm>
 92    <vm name="R6">
 93      <if id="1" net="f">
 94        <ipv4>10.0.6.2</ipv4>
 95      </if>
 96      <if id="2" net="g">
 97        <ipv4>10.0.7.2</ipv4>
 98      </if>
 99      <if id="3" net="h">
100        <ipv4>10.0.8.2</ipv4>
101      </if>
102      <exec seq="start" type="verbatim">zebra -f
     /etc/quagga/zebra.conf -d</exec>
103      <exec seq="start2" type="verbatim">ripd -s -f
     /etc/quagga/zebra.conf -x 5000 -d</exec>
104      <exec seq="stop" type="verbatim">killall zebra</exec>
105      <exec seq="stop2" type="verbatim">killall ripd</exec>
106    </vm>
107  </vnuml>
```

Quellcode F.4: XML-Datei für VNUML. Testsimulation 2 für Slow Start.

F.2 Testergebnisse der erwarteten Zeitdauer bei der Erkennung einer Schleife

In diesem Kapitel sind die kompletten Testergebnisse der Versuche aus Kapitel 4.2.2 aufgeführt. In den Tabellen ist in der ersten Spalte jeweils die laufende Nummer des Versuchs eingetragen. Die zweite Spalte gibt die gemessene Zeitdauer in Sekunden wieder, bis die Erreichbarkeitsinformationen zu Netzwerk *d* bei Router *R1* eingetroffen sind. Die Spalte „Durchschnittliche Zeitdauer“ gibt den Mittelwert der bis zu diesem Versuch gemessenen Zeiträume in Sekunden an. In der letzten Spalte ist der berechnete Erwartungswert angegeben.

Tabelle F.1 zeigt die Testergebnisse von Szenario 1, in Tabelle F.2 sind die Testergebnisse von Szenario 2 dargestellt.

Versuch	Gemessene Zeitdauer	Durchschnittliche Zeitdauer	Erwartungswert
1	23	23,0	16,8
2	31	27,0	16,8
3	25	26,3	16,8
4	14	23,3	16,8
5	20	22,6	16,8
6	4	19,5	16,8
7	15	18,9	16,8
8	31	20,4	16,8
9	16	19,9	16,8
10	10	18,9	16,8
11	14	18,5	16,8
12	21	18,7	16,8
13	3	17,5	16,8
14	17	17,4	16,8
15	12	17,1	16,8
16	30	17,9	16,8
17	10	17,4	16,8
18	4	16,7	16,8
19	37	17,7	16,8
20	24	18,0	16,8
21	39	19,0	16,8
22	28	19,4	16,8
23	4	18,8	16,8
24	40	19,7	16,8
25	34	20,2	16,8
26	33	20,7	16,8
27	7	20,2	16,8
28	15	20,0	16,8
29	8	19,6	16,8
30	22	19,7	16,8
31	21	19,7	16,8
32	13	19,5	16,8
33	13	19,3	16,8
34	32	19,7	16,8

Tabelle F.1: Testergebnisse bei Szenario 1.

Versuch	Gemessene Zeitdauer	Durchschnittliche Zeitdauer	Erwartungswert
1	47	47,0	15,1
2	31	39,0	15,1
3	18	32,0	15,1
4	20	29,0	15,1
5	27	28,6	15,1
6	24	27,8	15,1
7	8	25,0	15,1
8	14	23,6	15,1
9	12	22,3	15,1
10	22	22,3	15,1
11	23	22,3	15,1
12	13	21,6	15,1
13	9	20,6	15,1
14	10	19,9	15,1
15	6	18,9	15,1
16	26	19,4	15,1
17	11	18,9	15,1
18	11	18,4	15,1
19	28	18,9	15,1
20	9	18,4	15,1
21	15	18,2	15,1
22	14	18,0	15,1
23	17	18,0	15,1
24	11	17,8	15,1
25	11	17,5	15,1
26	11	17,2	15,1
27	26	17,6	15,1
28	17	17,5	15,1
29	16	17,5	15,1
30	9	17,2	15,1
31	35	17,8	15,1
32	17	17,8	15,1
33	17	17,7	15,1
34	36	18,3	15,1

Tabelle F.2: Testergebnisse bei Szenario 2.

F.3 Testergebnisse zu Slow Start

Folgend die Testergebnisse der Validierung von Slow Start. Um die Testergebnisse möglichst übersichtlich darzustellen, wurden die Grafiken auf das Notwendigste beschränkt. Die vollständigen Ergebnisse können auf der beiliegenden CD-ROM eingesehen werden.

Bei der Auswertung der Daten muss beachtet werden, dass die Zeitangaben in den Logdateien die Zeitbasis UTC verwenden, während die Zeitangaben in Wireshark auf der MEZ (UTC +1) basieren. Daher resultiert die Zeitdifferenz von einer Stunde.

Testfall 1: Zuerst die Ergebnisse der Tests bei deaktiviertem Slow Start. In den Abbildungen F.1 und F.2 sind zwei Update-Pakete vom gerade aktivierten Router *R6* (10.0.5.2/24) an seinen Nachbarrouter *R5* (10.0.5.1/24) über Netzwerk *e* (10.0.5.0/24) zu sehen.

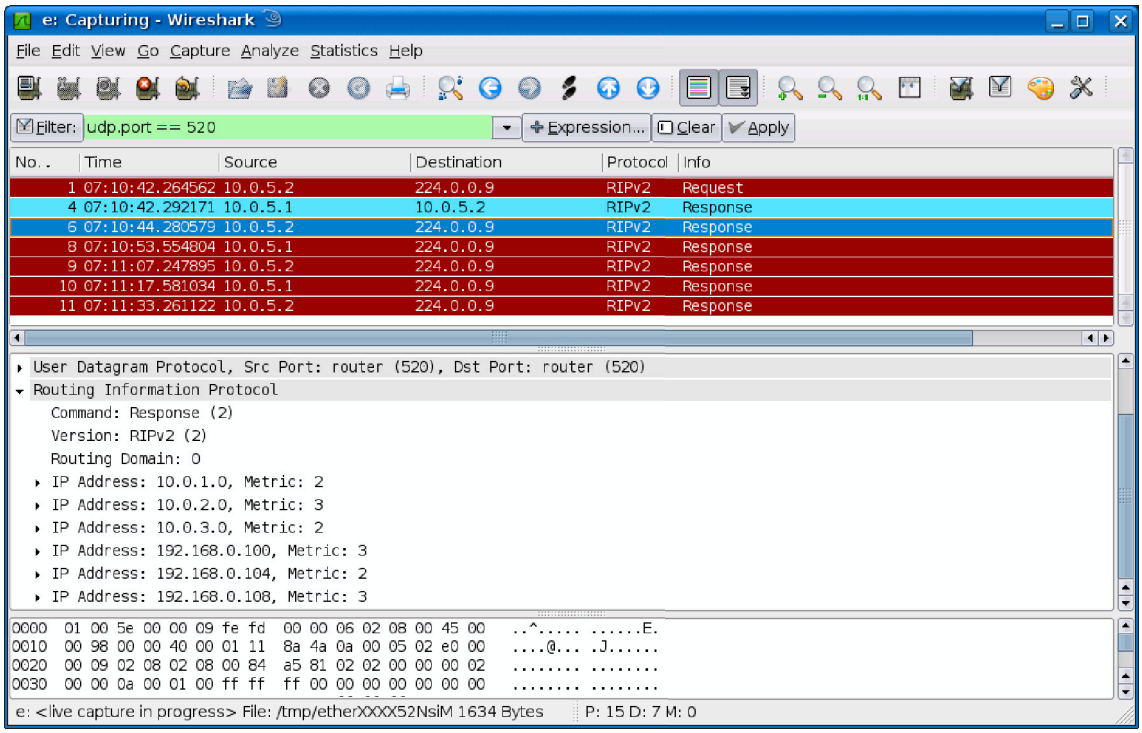


Abbildung F.1: Testfall 1, Slow Start inaktiv, Netzwerk e, erste Update-Nachricht von R6.

In Abbildung F.1 sind im oberen Teil die einzelnen Nachrichten zwischen den Routern sichtbar. Zuerst hat *R6* ein Request an *R5* geschickt, der dieses auch umgehend beantwortet hat. Das anschließende Response von *R6* ist in Abbildung F.1 ausführlich dargestellt. Die Netzwerke *a* (10.0.1.0/24) und *c* (10.0.3.0/24) werden mit der Metrik 2 an *R5* gemeldet, das Netzwerk *b* (10.0.2.0/24) mit der Metrik 3. Das direkt an *R6* angeschlos-

sene Netzwerk *d* (10.0.4.0/24) fehlt in dieser Update-Nachricht. Warum dies so ist, zeigt später die Logdatei F.1.

Bei den Responses mit den laufenden Nummern 8 und 10 handelt es sich um Timed Updates von *R5*.

Response Nummer 9 ist das erste Timed Update von *R6* (Abbildung F.2). Es ist die erste Update-Nachricht von *R6*, in der das direkt angeschlossene Netzwerk *d* an *R5* gemeldet wird. Somit sind mit dieser Nachricht *R5* alle über *R6* erreichbaren Netzwerke bekannt.

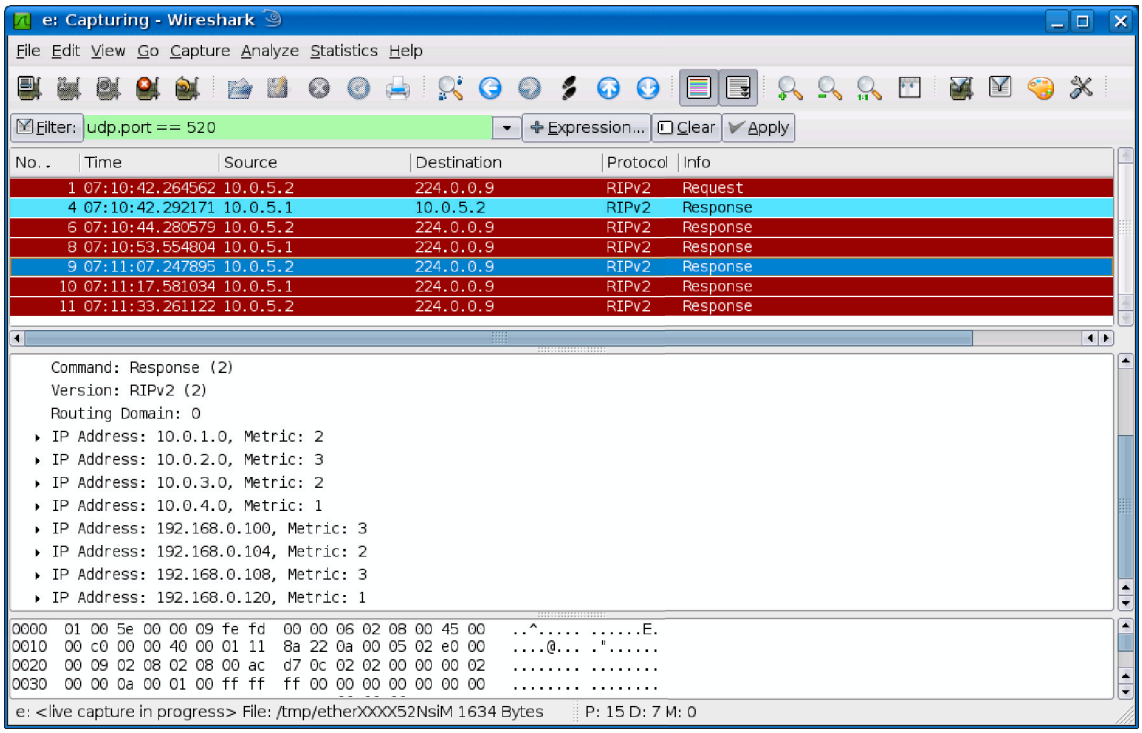


Abbildung F.2: Testfall 1, Slow Start inaktiv, Netzwerk e, zweite Update-Nachricht von R6.

Warum *R6* erst mit dem ersten Timed Update das direkt angeschlossene Netzwerk *d* an *R5* meldet, ist aus Logdatei F.1 ersichtlich. Nachdem in Zeile `4` das erste Interface mit dem dahinter liegenden Netzwerk erkannt wurde, versucht der Router ein Triggered Update zu versenden (Zeile `6`). Es wird jedoch zum einen wegen Split Horizon nicht gesendet und zum anderen, weil noch kein Request verschickt wurde. In Zeile `18`, nachdem alle Interfaces mit den dahinter liegenden Netzwerken erkannt wurden und das erste Request auf dem ersten Interface verschickt wurde, versendet der Router erneut ein Triggered Update mit Informationen zu den direkt angeschlossenen Netzwerken. Weil allerdings bislang nur das eine Request verschickt wurde, wird das Triggered Update auch nur auf diesem einen Interface versendet. Da das Triggered Update erfolgreich verschickt wird, sind die Routen zu den Netzwerken *d* und *e* in der Routingtabelle nicht mehr als „geändert“ markiert und werden deshalb nach dem Versand der Requests

auf den anderen Interfaces nicht mehr erneut versendet, sondern erst mit dem ersten Timed Update.

Bei VNUML handelt es sich bei dem ersten Interface um das Management Interface der VM. Bei einem realen Router würde der Start aber genauso ablaufen, so dass die Informationen zu den direkt angeschlossenen Netzwerken nur über ein einziges Interfaces per Triggered Update verbreitet werden und die restlichen Interfaces von diesen Netzwerken erst mit dem ersten Timed Update erfahren.

```
  4  06:10:41 RIP: turn on eth0
  6  06:10:41 RIP: triggered update!
  8  06:10:41 RIP: turn on eth1
 11  06:10:41 RIP: turn on eth2
 18  06:10:42 RIP: multicast request on eth0
 21  06:10:42 RIP: triggered update!
 22  06:10:42 RIP: SEND UPDATE to eth0 ifindex 3
 32  06:10:42 RIP: multicast request on eth1
 37  06:10:42 RIP: multicast request on eth2
 42  06:10:42 RIP: RECV packet from 10.0.4.1 port 520 on eth1
 50  06:10:42 RIP: RECV packet from 10.0.5.1 port 520 on eth2
 55  06:10:44 RIP: triggered update!
 56  06:10:44 RIP: SEND UPDATE to eth0 ifindex 3
 71  06:10:44 RIP: SEND UPDATE to eth1 ifindex 1
 80  06:10:44 RIP: SEND UPDATE to eth2 ifindex 2
 95  06:10:53 RIP: RECV packet from 10.0.5.1 port 520 on eth2
100  06:11:03 RIP: RECV packet from 10.0.4.1 port 520 on eth1
108  06:11:07 RIP: update timer fire!
109  06:11:07 RIP: SEND UPDATE to eth0 ifindex 3
126  06:11:07 RIP: SEND UPDATE to eth1 ifindex 1
137  06:11:07 RIP: SEND UPDATE to eth2 ifindex 2
```

Logdatei F.1: Auszug aus der Logdatei von R6 ohne Slow Start.

Ist Slow Start aktiviert läuft die Kommunikation zwischen *R5* und *R6* wie in Abbildung F.3 dargestellt ab. Nach dem initialen Request schickt *R6* erst mit dem ersten Timed Update seine komplette Routingtabelle an *R5*.

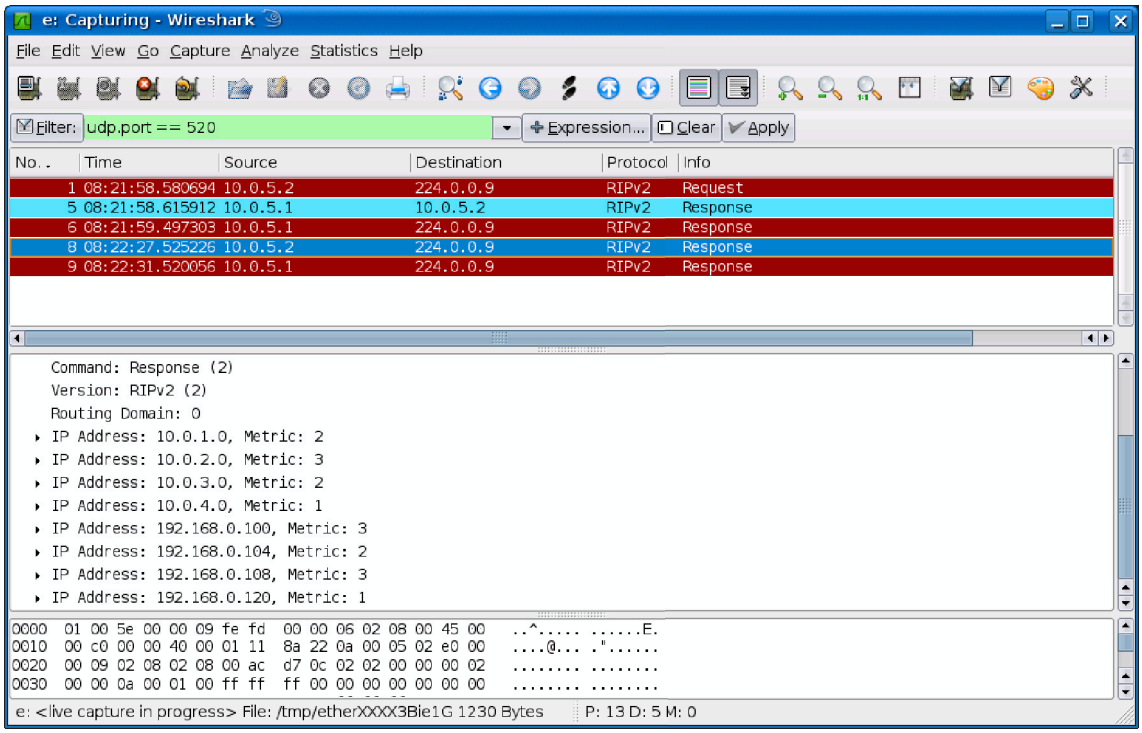


Abbildung F.3: Testfall 1, Slow Start aktiv, Netzwerk e, Update-Nachricht von R6.

In Logdatei F.2 sind deutlich die unterdrückten Triggered Updates zu erkennen. Der Router versucht hier wesentlich öfter ein Triggered Update zu versenden, da wegen dem Abbruch des Versandes der Timer nicht gestartet wird.
In Zeile 63 ist der Wechsel von der Startphase in die Betriebsphase ersichtlich. Direkt anschließend wird das erste Timed Update verschickt.

```
 2  07:21:57 RIP: SlowStart enabled!
 5  07:21:57 RIP: turn on eth0
 7  07:21:57 RIP: SlowStart triggered update canceled!
 9  07:21:57 RIP: turn on eth1
11  07:21:57 RIP: SlowStart triggered update canceled!
13  07:21:57 RIP: turn on eth2
15  07:21:57 RIP: SlowStart triggered update canceled!
21  07:21:58 RIP: multicast request on eth0
25  07:21:58 RIP: multicast request on eth1
31  07:21:58 RIP: multicast request on eth2
34  07:21:58 RIP: RECV packet from 10.0.4.1 port 520 on eth1
42  07:21:58 RIP: SlowStart triggered update canceled!
44  07:21:58 RIP: RECV packet from 10.0.5.1 port 520 on eth2
49  07:21:58 RIP: SlowStart triggered update canceled!
50  07:21:59 RIP: RECV packet from 10.0.5.1 port 520 on eth2
55  07:22:25 RIP: RECV packet from 10.0.4.1 port 520 on eth1
63  07:22:27 RIP: SlowStart now in running mode
64  07:22:27 RIP: update timer fire!
```

```
65  07:22:27 RIP: SEND UPDATE to eth0 ifindex 3
82  07:22:27 RIP: SEND UPDATE to eth1 ifindex 1
95  07:22:27 RIP: SEND UPDATE to eth2 ifindex 2
```

Logdatei F.2: Auszug aus der Logdatei von R6 bei aktivem Slow Start.

Testfall 2: Der Nachrichtenversand bei diesem Testfall ist sehr ähnlich zu dem zuvor beschriebenen Testfall. In Abbildung F.4 und Abbildung F.5 sind die beiden von *R6* (10.0.8.2/24) ohne Slow Start verschickten Update-Nachrichten genau aufgeführt. Wie zuvor erfolgt die Übermittlung der direkt angeschlossenen Netzwerke *f* (10.0.6.0/24) und *g* (10.0.7.0/24) erst mit dem ersten Timed Update (Abbildung F.5).

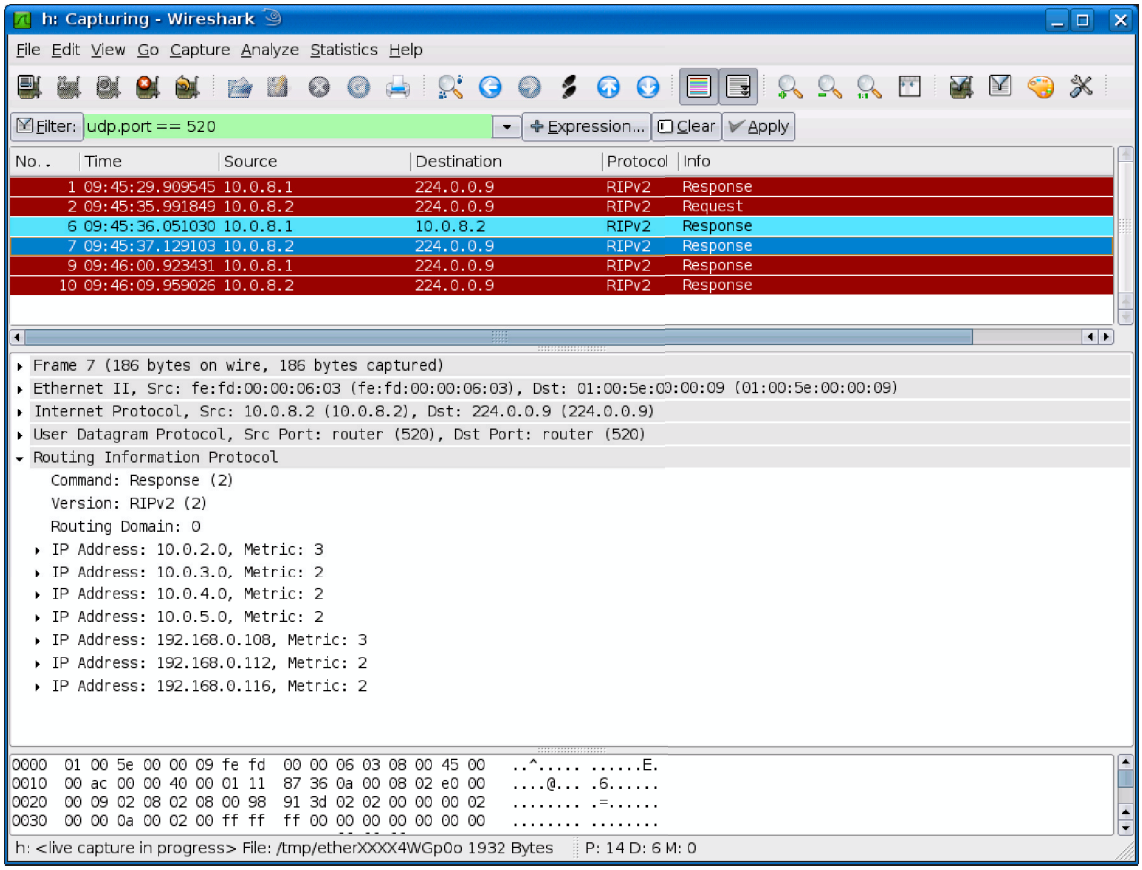


Abbildung F.4: Testfall 2, Slow Start inaktiv, Netzwerk h, erste Update-Nachricht von R6.

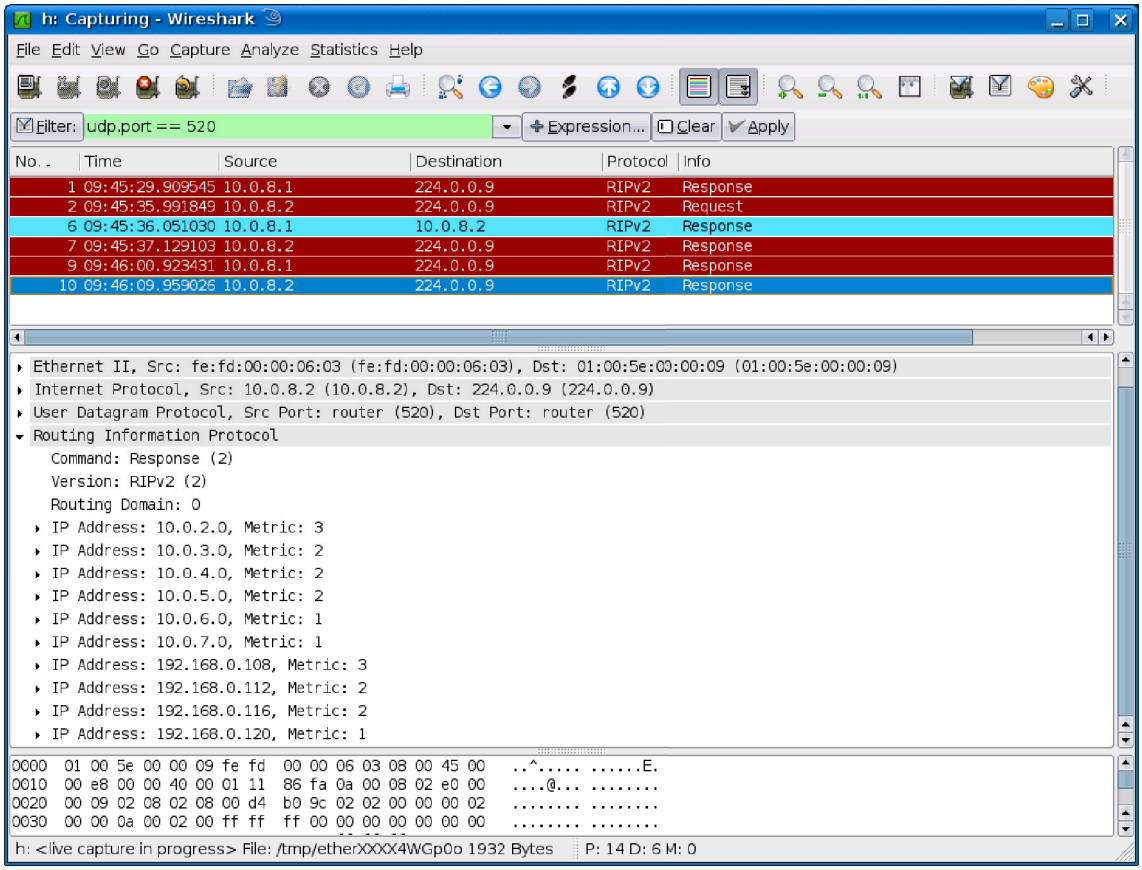


Abbildung F.5: Testfall 2, Slow Start inaktiv, Netzwerk h, zweite Update-Nachricht von R6.

In Logdatei F.3 von Router *R6* ist, wie beim ersten Testfall, der Grund für die späte Übermittlung der Erreichbarkeiten von den Netzwerken *f* und *g* ersichtlich.

```
  4  08:45:34 RIP: turn on eth0
  6  08:45:34 RIP: triggered update!
  8  08:45:34 RIP: turn on eth1
 11  08:45:34 RIP: turn on eth2
 14  08:45:34 RIP: turn on eth3
 21  08:45:35 RIP: multicast request on eth0
 24  08:45:35 RIP: triggered update!
 25  08:45:35 RIP: SEND UPDATE to eth0 ifindex 4
 36  08:45:35 RIP: multicast request on eth1
 41  08:45:35 RIP: multicast request on eth2
 46  08:45:35 RIP: multicast request on eth3
 51  08:45:36 RIP: RECV packet from 10.0.7.1 port 520 on eth2
 65  08:45:36 RIP: RECV packet from 10.0.6.1 port 520 on eth1
 79  08:45:36 RIP: RECV packet from 10.0.8.1 port 520 on eth3
 93  08:45:37 RIP: triggered update!
 94  08:45:37 RIP: SEND UPDATE to eth0 ifindex 4
110  08:45:37 RIP: SEND UPDATE to eth1 ifindex 1
124  08:45:37 RIP: SEND UPDATE to eth2 ifindex 2
135  08:45:37 RIP: SEND UPDATE to eth3 ifindex 3
152  08:45:47 RIP: RECV packet from 10.0.6.1 port 520 on eth1
164  08:45:49 RIP: RECV packet from 10.0.7.1 port 520 on eth2
177  08:46:00 RIP: RECV packet from 10.0.8.1 port 520 on eth3
```

```
187  08:46:09 RIP: update timer fire!
188  08:46:09 RIP: SEND UPDATE to eth0 ifindex 4
207  08:46:09 RIP: SEND UPDATE to eth1 ifindex 1
224  08:46:09 RIP: SEND UPDATE to eth2 ifindex 2
238  08:46:09 RIP: SEND UPDATE to eth3 ifindex 3
```

Logdatei F.3: Auszug aus der Logdatei von R6 ohne Slow Start.

Mit Slow Start ist der Ablauf in diesem Testfall dem Ablauf aus Testfall 1 mit Slow Start sehr ähnlich. In Abbildung F.6 ist die erste Update-Nachricht ausführlich dargestellt. Die unterdrückten Triggered Updates sowie der Wechsel zur Betriebsphase können in Logdatei F.4 nachvollzogen werden.

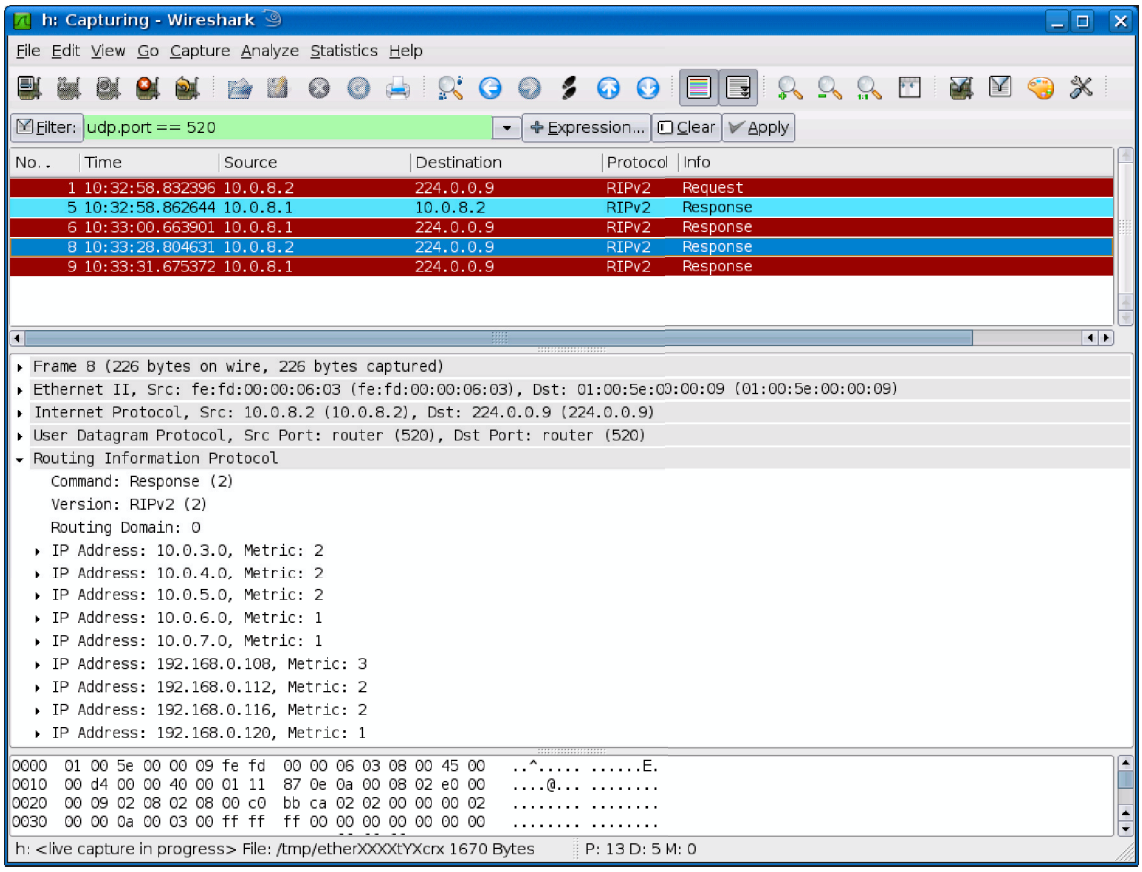


Abbildung F.6: Testfall 2, Slow Start aktiv, Netzwerk h, Update-Nachricht von R6.

```
2   09:32:57 RIP: SlowStart enabled!
5   09:32:57 RIP: turn on eth0
7   09:32:57 RIP: SlowStart triggered update canceled!
9   09:32:57 RIP: turn on eth1
11  09:32:57 RIP: SlowStart triggered update canceled!
13  09:32:57 RIP: turn on eth2
15  09:32:57 RIP: SlowStart triggered update canceled!
17  09:32:57 RIP: turn on eth3
19  09:32:57 RIP: SlowStart triggered update canceled!
25  09:32:58 RIP: multicast request on eth0
29  09:32:58 RIP: multicast request on eth1
34  09:32:58 RIP: multicast request on eth2
39  09:32:58 RIP: multicast request on eth3
```

```
 44  09:32:58 RIP: RECV packet from 10.0.8.1 port 520 on eth3
 58  09:32:58 RIP: SlowStart triggered update canceled!
 59  09:32:58 RIP: RECV packet from 10.0.6.1 port 520 on eth1
 73  09:32:58 RIP: SlowStart triggered update canceled!
 74  09:32:58 RIP: RECV packet from 10.0.7.1 port 520 on eth2
 88  09:32:58 RIP: SlowStart triggered update canceled!
 89  09:33:00 RIP: RECV packet from 10.0.6.1 port 520 on eth1
103  09:33:00 RIP: RECV packet from 10.0.8.1 port 520 on eth3
117  09:33:24 RIP: RECV packet from 10.0.7.1 port 520 on eth2
131  09:33:28 RIP: SlowStart now in running mode
132  09:33:28 RIP: update timer fire!
133  09:33:28 RIP: SEND UPDATE to eth0 ifindex 4
152  09:33:28 RIP: SEND UPDATE to eth1 ifindex 1
168  09:33:28 RIP: SEND UPDATE to eth2 ifindex 2
184  09:33:28 RIP: SEND UPDATE to eth3 ifindex 3
```

Logdatei F.4: Auszug aus der Logdatei von R6 bei aktivem Slow Start.

Anhang G CD-ROM

Die beiliegende CD-ROM auf der hinteren Umschlaginnenseite enthält folgende Dateien und Verzeichnisse:

CD-ROM-Laufwerk	
code	Der Order „code" enthält den gesamten Programmcode.
quagga-0.99.11.tar.gz	Dieses Archiv enthält den originalen Quellcode des Quagga, an dem Änderungen vorgenommen wurden.
rip.tar.gz	Das Archiv enthält den Quellcode der Implementierung von Slow Start in RIP
rip-mti.tar.gz	Das Archiv enthält den Quellcode der Implementierung von Slow Start in RIP-MTI
rip-patch.tar.gz	In diesem Archiv befindet sich ein Patch mit den Unterschieden zwischen dem originalen Quagga und der darauf basierenden Slow Start Implementierung.
performanz_von_rip-mti.pdf	Diese Diplomarbeit liegt als elektronisches Dokument für den Adobe Reader[12] vor.
quellen	Der Ordner „quellen" enthält die elektronischen Quellen im Adobe PDF-Format.
testdaten_slowstart	Dieser Ordner enthält Logdateien und Screenshots der beiden Testszenarien von Slow Start.
vnuml	Der Ordner „vnuml" enthält die für VNUML benötigten Dateien.
simulationen	Der Ordner „simulationen" enthält alle in dieser Arbeit benutzten VNUML-Simulationen.
mini_vnuml-1.8-quagga.img	Das VNUML Mini-Image enthält eine Installation von RIP mit Slow Start.
mini_vnuml-1.8-ripmti.img	Dieses VNUML Mini-Image enthält eine Installation von RIP-MTI mit Slow Start.
VNUML-Offline-1.8.3-1.tar.gz	Dieses Archiv enthält den VNUML-Offline-Installer zur Installation von VNUML.
xtpeer.jar	Die Datei enthält den verwendeten XTPeer.

12 Der Adobe Reader ist ein Programm zur Anzeige von PDF-Dokumenten und kann unter http://get.adobe.com/de/reader/ kostenlos runtergeladen werden.