



**Telecooperation Office (TecO)
Institut für Telematik
Universität Karlsruhe**

Studienarbeit

Routing in dynamic, arbitrarily and partially connected networks

Christian Decker

Betreuer: Prof. Dr. Lars Wolf
Betreuende Mitarbeiter: Albrecht Schmidt, Michael Beigl

Anmeldung: 01.07.2001
Abgabe: 30.09.2001

Inhaltsverzeichnis

1	Einleitung.....	1
2	Routing – Grundlagen im Zusammenhang mit FAN.....	3
2.1	Routing - Allgemein.....	3
2.2	Routing Algorithmen - Design Ziele.....	3
2.3	Routing Algorithmen - Überblick.....	3
2.4	Routing Algorithmen im FAN.....	5
2.4.1	Analyse der bisher vorgestellten Routing Algorithmen.....	5
2.4.2	Eingesetzte Routing Algorithmen.....	6
3	Simulation und Implementierung.....	7
3.1	FAN Simulations Architektur.....	7
3.1.1	Architektur - Überblick.....	7
3.1.2	Simulator und seine Komponenten.....	7
3.2	Implementierung der FAN Simulations Architektur.....	9
3.2.1	Vergleich der Umgebungen ns-2 und Ptolemy II.....	9
3.2.2	Implementierungsdetails.....	10
3.2.3	Einschränkungen und zukünftige Arbeiten.....	12
4	Routing – Simulationsergebnisse.....	13
4.1	Untersuchung des Ressourcenbedarfs.....	13
4.1.1	Simulation - Topologie und Parameter.....	13
4.1.2	Simulation - Ergebnisse und Auswertung.....	14
4.1.3	Simulation - Zusammenfassung.....	18
4.2	Untersuchung der Netzwerkdynamik.....	18
4.2.1	Simulation - Topologie und Parameter.....	19
4.2.2	Simulation - Ergebnisse und Auswertung.....	20
4.2.3	Simulation - Zusammenfassung.....	29
5	Zusammenfassung und Ausblick.....	30
6	Literaturverzeichnis.....	31

1 Einleitung

Wir erleben derzeit eine Explosion des Einsatzes von persönlichen Geräten. Viele davon werden herumgetragen, manche sind gar an Körper oder Kleidung angebracht. Heutzutage arbeiten diese Geräte unabhängig voneinander ohne die Möglichkeit einer Kommunikation nutzen zu können. Dies wird sich mit dem Einsatz ubiquitärer Kommunikations- und Computersysteme [Weiser, 1991] ändern. Bereits bestehende Anwendungen werden dann aus den neu zur Verfügung stehenden Information zusätzlichen Nutzen ziehen können. Durch den Einsatz dieser neuen Technologie werden aber auch ganz neue Anwendungen entstehen. Die Vernetzung solch kleiner Geräten ist das Ziel von mehreren neuartigen technologischen Entwicklungen, etwa von funkbasierten Netzwerken mit geringer Reichweite (Pico-Netzwerken) wie Bluetooth [Sonnerstam, 1999] und auch Netzwerke, die am Körper getragen werden. Die genannten Systeme haben allerdings den Nachteil, dass sie die Informationen in die nähere Umgebung senden [Padridge et al., 2000] und damit verwundbar sind für potentielle Angreifer. Ausserdem verbrauchen sie deutlich mehr Energie als drahtgebundene Ansätze

Diese Studienarbeit untersucht eine spezielle Art von Netzwerken, die Geräte miteinander verbinden, die am Körper getragen werden. Das sogenannte Fabric-Area-Network (FAN) [Hum, 2001] ist ein in die Kleidung eingebettetes drahtgebundenes Netzwerk, das einen geschützten Datentransport zwischen allen Geräten, die eine Verbindung zu Kleidungsstücken besitzen, ermöglicht. Anwendungsbereiche solcher Netzwerke sind Kommunikation von in Kleidung eingebetteten Sensoren mit einem Zentralcomputer, Anwendungen im Gesundheitsbereich (Herzschrittmacher, Uhren mit Vitalüberwachungsfunktionen etc.) oder Arbeits- und Schutzkleidung mit Pagern, Scannern und Spezialgeräten.

Innerhalb der Arbeit werden ein Prototyp-System präsentiert sowie Simulationen für Netzwerke, die in Kleidung implementiert sind, durchgeführt. Dabei liegt der Fokus auf dem Problem des Vermittelns und Durchleitens (Routen) von Datenpaketen innerhalb solcher Netzwerke. Übergänge von einem Kleidungsstück zu einem anderen sind Kreuzungspunkte von Verbindungen zwischen verschiedenen Teilen des FAN und bieten sich deshalb als Punkte für das Routen und Filtern an. Der Einsatz von Routern ermöglicht es, die Auslastung und den Energieverbrauch von Stationen zu steuern und damit auch die Auslastung des gesamten Netzwerkes zu beeinflussen. Die Möglichkeit, zu entscheiden, welche Wege Pakete an Kreuzungspunkten einschlagen, sollte die Leistungsfähigkeit des Netzwerkes steigern und es an die Dynamik der Umgebung, in der es eingesetzt wird, besser anpassen.

Im Anschluss an diese Einleitung wird ein detaillierter Einblick in die FAN-Technologie gegeben. Hier werden dann Rahmenbedingungen für spätere Simulationen festgelegt. Es folgt im darauffolgenden Abschnitt eine Betrachtung verschiedener Routing-Verfahren und eine Analyse für ihren Einsatz im Fabric-Area-Network. Ergebnis dieser Untersuchungen ist eine Auswahl von Routing Protokollen, deren Verhalten in solchen Netzwerken untersucht werden soll. Dazu wird im nächsten Abschnitt eine Architektur für FAN-Simulationen entworfen und implementiert. Diese dient als Testumgebung für die daran angeschlossenen Untersuchungen des Verhaltens und der Leistungsfähigkeit der Protokolle. Besonderes Augenmerk wird dabei auf den Ressourcenverbrauch und das Verhalten in verschiedenen, dynamisch generierten Netzwerksituationen eines jeden Protokolls gelegt. Für vergleichende Aussagen wurden Metriken wie Verlustrate und Anzahl der Hops herangezogen. Die Auswertung der Simulationsergebnisse wird im letzten Abschnitt zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

FAN im Detail

Das Fabric-Area-Network (FAN) ist eine Architektur, die verschiedene Kleidungsstücke zum Zweck des Datenaustausches miteinander verbindet. Kleidungsstücke sollen dabei nicht fest miteinander verbunden werden, um ihren täglichen Gebrauch nicht einzuschränken. FAN verwendet dazu eine Kopplung auf Basis eines eng begrenzten elektromagnetischen Feldes [Hum, 2001]. Dabei bilden zwei Spulen ein einfaches Sender/Empfänger System, zwischen denen mittels Modulation des elektromagnetischen Feldes Daten ausgetauscht werden

können. In Abbildung 1 wird eine Verbindung eines T-Shirt mit einer Hose über FAN schematisch gezeigt.

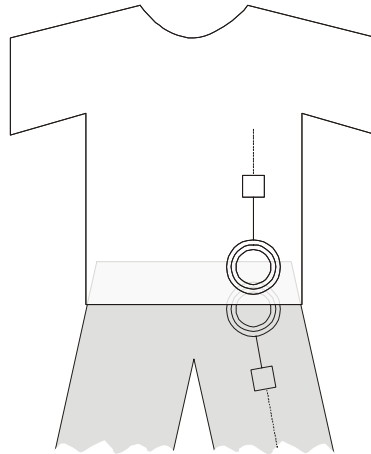


Abbildung 1 FAN Link

Über solche Links werden Daten in 18 byte Paketen mit 1000 bits/s übertragen. Aber bevor dies geschehen kann, baut die Treiberhardware des Senders das elektromagnetische Feld auf und initiiert ein Handshake-Verfahren, um den Empfänger zu detektieren. Dies beansprucht eine zusätzliche Latenzzeit von ca. 100 ms pro Datenpaket.

Datenverbindungen über elektromagnetische Kopplung verhalten sich nicht wie drahtgebundene. Solche Verbindungen wie sie in Abbildung 1 dargestellt sind, sind stark von der Bewegung des Trägers und äusseren Einflüssen abhängig. So ist es möglich, dass Verbindungen für einen gewissen Zeitraum abbrechen und dann wiederhergestellt werden. Aber selbst nach einer solchen Wiederherstellung ist es nicht sicher, dass die Senderspule an die selbe Empfängerspule gekoppelt ist wie vor dem Abbruch.

Ein weiterer Betrachtungspunkt ist die Treiberhardware. Sie nimmt Aufgaben wahr, wie das Aufbauen des elektromagnetischen Feldes, die Detektion der Empfängerspule, Aushandeln des Senders bzw. Empfängers bei Halb-Duplex Betrieb und Fehlerkorrektur der Pakete über ein CRC Verfahren. Diese Hardware ist sehr klein, da sie mehrmals auf den verschiedenen Kleidungsstücken angebracht wird und dabei aber das Tragen nicht beeinflussen soll. Daraus resultiert, dass nur sehr begrenzte Ressourcen wie Rechenleistung, Speicher und Energie vorhanden sind. Insbesondere ist der Energieverbrauch sehr kritisch, da fast die gesamte zur Verfügung stehende Energie für die Versorgung des elektromagnetischen Feldes benötigt wird.

Das Fabric-Area-Network arbeitet in einer sehr dynamischen Umgebung. Verbindungen lassen nur geringe Bandbreiten zu, unterliegen der Dynamik der Kleidungsstücke zwischen denen sie angebracht sind und sind in hohem Maße unzuverlässig. Die Treiberhardware muss die an sie gestellten Anforderungen mit sehr wenig Ressourcen bewältigen.

2 Routing - Grundlagen im Zusammenhang mit FAN

Dieser Abschnitt befasst sich mit den grundlegenden Konzepten in weitverbreiteten Routing Protokollen. Dabei wird kurz auf unterschiedliche Protokolle eingegangen und im Anschluss ihre Verwendbarkeit für FAN untersucht. Ziel dieses Abschnittes ist es in FAN einsetzbare Protokolle zu bestimmen, die für die weitere Arbeit in Simulationen verwendet werden.

2.1 Routing - Allgemein

Unter Routing wird die Durchleitung von Daten durch ein Netzwerk von einer Quelle zu einem oder mehreren Zielen verstanden. Dabei werden zwei grundlegende Aktionen ausgeführt. Zum ersten wird der optimale Weg festgestellt und zum zweiten werden die Daten meistens in Form von Paketen durch das Netzwerk transportiert. Letzteres bezeichnet man auch oft mit dem Begriff *Switching*. Während das Switching recht einfach ist, kann der Prozess der Wegbestimmung sehr komplex werden.

2.2 Routing Algorithmen - Design Ziele

Die Wegberechnung beim Routing wird von Routing Algorithmen übernommen. Diese basieren auf verschiedene Charakteristiken, beeinflussen den Datentransport auf verschiedene Weise und benutzen verschiedene Messeinheiten (Metriken). Alle verfolgen dabei ein oder mehrere der folgenden Ziele:

- optimale Ergebnisse liefern
- einfach und wenig Mehrarbeit (overhead)
- robust und stabil
- schnelle Konvergenz zum Optimum
- flexibel

Aufgabe eines Routing Algorithmus ist es, eine optimale Route zu liefern, indem er unter Verwendung von Metriken die beste Route auswählt. Viele Algorithmen sind ausserdem darauf ausgerichtet, so einfach wie möglich zu sein. Dabei sollen sie effizient funktionieren mit einem Minimum an Software und Mehrarbeit für das System, auf dem er läuft. Das ist im Besonderen wichtig für Algorithmen in Netzwerken mit sehr begrenzten Ressourcen wie das in dieser Arbeit betrachtete FAN. Ein ebenfalls sehr wichtiges Ziel ist die Robustheit. Ein Algorithmus soll auch korrekt arbeiten, wenn unvorhergesehene Umstände eintreten, wie Hardwareversagen oder hohe Last. Er soll stabil sein, d.h. weiterhin optimale Ergebnisse liefern, unter verschiedenen Bedingungen im Netzwerk. Schnelle Konvergenz heisst für einen Algorithmus, dass er schnell den optimalen Weg bestimmen kann. Langsame Konvergenz führt zu Fehlentscheidungen beim Routing, die verursachen, dass Pakete immer wieder bei ein und demselben Router vorbeikommen (sogenannte *loops*) oder dass Teile des Netzwerkes nicht mehr erreicht werden. Algorithmen werden auch mit dem Ziel flexibel zu sein entworfen. Damit meint man, dass sie sich schnell und richtig auf verschiedene Umstände im Netzwerk einstellen können.

Neben oben aufgezählten Zielen sind noch andere vorstellbar wie Korrektheit oder auch Gesichtspunkte aus dem Bereich von Quality of Service (QoS).

2.3 Routing Algorithmen - Überblick

Es ist wünschenswert, dass ein Routing Algorithmus möglichst viele der obengenannten Ziele sehr gut erfüllt. In der Praxis ist das aber oftmals nicht möglich, weswegen es viele Algorithmen gibt, die zwar sehr gut sind, aber nur unter bestimmten Bedingungen. Im Folgenden werden einige Klassen von Algorithmen aufgeführt und in ihren Eigenschaften beschrieben.

Zu beschreibende Klassen von Routing Algorithmen sind:

- Statisches Routing
- Quellenbasiertes Routing
- Link State Routing
- Distance Vector Routing

Statisches Routing

Beim statischen Routing sind die Wege, die Datenpakete von der Quelle zum Ziel verfolgen, vom Administrator eines Netzwerkes in Form von Routing Tabellen noch vor dem Beginn des Routing vorgegeben. Die Einträge der Tabellen ändern sich solange nicht, bis sie der Administrator ändert. Algorithmen, die statische Routen benutzen, sind sehr einfach und funktionieren gut in Netzwerken, deren Struktur einfach ist und in denen der Datenverkehr gut vorhersagbar ist.

Quellenbasiertes Routing

Bei dieser Art kennt die Quelle des Datenpaketes den Weg zu dessen Ziel. Deshalb werden solche Algorithmen auch als *source routing* Algorithmen bezeichnet. Die Route wird dabei in dem Paket selbst hinterlegt. Transit-Router funktionieren in solchen Netzwerken als einfache Store-and-Forward Geräte. In solchen Netzwerken werden sehr gute, sprich optimale, Routen gefunden, da die Quellsysteme typischerweise alle möglichen Routen entdecken, bevor das Paket gesendet wird. Dieser Entdeckungsprozess nimmt aber eine gewisse Zeit in Anspruch und verursacht weiteren Netzwerkverkehr. Source Routing wird als dynamisches Source Routing gern in drahtlosen Netzwerken eingesetzt. [Johnson & Maltz, 1996] untersuchten diesen Algorithmus mit einer Reihe von Optimierungen.

Die folgenden zwei Routing-Algorithmen sind dynamische routerbasierte Algorithmen. Das heisst, dass die Wege nicht vorgegeben werden, sondern von jedem Router selbst bestimmt werden. Im Gegensatz zu den statischen Algorithmen, können dynamische selbständig auf Veränderungen im Netzwerk reagieren und sind damit für grosse, dynamische Netzwerke interessant. Für die Bestimmung der Routen wird eine Messeinheit, auch Metrik genannt, verwendet. Die gebräuchlichste ist dabei die Weglänge in Form von Hops, welche die Anzahl der auf dem Weg zurückgelegten Stationen darstellt. Aber auch andere wie zum Beispiel Kosten, Zuverlässigkeit oder auch Verwaltung der verschiedenen Teilstrecken finden Verwendung. Zur Unterstützung werden von Routing Algorithmen oftmals Routing Tabellen angelegt, mit denen solche Informationen über Metriken verwaltet werden können. Erhalten werden diese Informationen durch eigene Messungen des Routers, auf dem das Routing Protokoll läuft, aber auch durch Austausch von ganzen Routing Tabellen oder nur Teilen davon mit anderen Routern.

Link State Routing

Diese Algorithmen sind auch als *shortest path first* Algorithmen bekannt. Jeder Router sendet dazu den Teil seiner Routing Tabelle, der den Status seiner Verbindungen (link) beschreibt zu allen anderen Routern im Netzwerk. Das wird auch als Fluten (flooding) bezeichnet. Als Ergebnis hat jeder Router eine Gesamtübersicht über das gesamte Netzwerk durch seine Routing Tabelle. Damit kann dann mittels eines Shortest-Path Algorithmus in Abhängigkeit zu den gewählten Metriken der kürzeste Weg zum Ziel bestimmt werden. Jeder Router bestimmt dabei für jedes mögliche Ziel den kürzesten Weg mit sich selbst als Ausgangspunkt. Link State Algorithmen konvergieren sehr schnell, brauchen aber aufgrund ihrer Komplexität viel Rechenleistung und vor allem viel Speicher für die Routing Tabellen. Ein bekannter Vertreter dieser Klasse ist Open Shortest Path First (OSPF)[Moy, 1998].

Distance Vector Routing

Diese Algorithmen beziehen ihre Informationen über mögliche Wege zum Ziel von den Nachbar-Routern. Diese werden aufgefordert, Teile ihrer Routing Tabelle zu übertragen. Damit wird eine Übersicht über das Netzwerk gebildet, ohne das Netzwerk zu fluten. Durch das Einfügen einer fremden Routing Tabelle in die eigene kann ausserdem der Speicherbedarf optimiert werden. Weiterhin benötigen Berechnungen über eine Route zum Ziel weniger Rechenleistung, da auf die Berechnungen der Nachbarn zurückgegriffen wird. Allerdings konvergieren diese Algorithmen langsamer als Link State Algorithmen, da die Netzwerkübersicht nur über die Kommunikation mit den Nachbarroutern entsteht. Ein bekannter Vertreter dieser Klasse ist das Routing Information Protocol (RIP)[Malkin, 1998].

2.4 Routing Algorithmen im FAN

Nachdem ein Überblick über verschiedene Routing Algorithmen gegeben wurde, werden diese nun hinsichtlich ihrer Verwendbarkeit im FAN untersucht. Gemäss den Detail-Betrachtungen in der Einleitung müssen Routing Algorithmen dabei mit einem sehr dynamischen Netzwerk umgehen, in dem die Verbindungen nur geringe Bandbreiten zulassen und unzuverlässig sind. Ausserdem wird die Leistung stark von den limitierten Ressourcen der Treiberhardware begrenzt.

2.4.1 Analyse der bisher vorgestellten Routing Algorithmen

Für den Einsatz in FAN ist statisches Routing sehr schlecht geeignet. Wie bereits festgestellt wurde, ist dieses Netzwerk sehr dynamisch. Allein durch einfache Bewegung des Trägers der Kleidung kann sich die Topologie und damit auch mögliche Routen ändern. Weiterhin können durch die Kopplung über elektromagnetische Spulen jederzeit neue Geräte an das Netz angeschlossen werden bzw. von diesem entfernt werden. Dabei ist der Ort der Ankopplung nicht festgelegt, sondern frei wählbar. Ein statischer Routing Algorithmus müsste daraufhin immer wieder neu eingestellt werden. Für FAN ist also nur ein dynamischer Routing Algorithmus geeignet.

Ein quellenbasierter Routing Algorithmus könnte mit dem Problem der Netzwerkdynamik umgehen. Problem ist aber die Mehrarbeit, die verrichtet werden muss, um eine geeignete Route zu entdecken. Aufgrund der geringen Bandbreite benötigt diese Entdeckung sehr lange bis das Paket abgeschickt werden kann. Die Unzuverlässigkeit trägt dazu bei, dass mehrere der Entdeckungspakete abgesetzt werden müssen (bis hin zum Fluten des Netzes), um eine Route zu ermitteln. Dies benötigt aber wiederum mehr Speicher und auch Rechenleistung zur Verwaltung des Entdeckungsmechanismus. Das resultiert in einer komplexeren Implementierung. Ein für FAN geeigneter Routing Algorithmus sollte die Entscheidung über die zu wählende Route vor Ort, d.h. in jedem zwischen Quelle und Ziel liegenden Router treffen.

Link State Algorithmen sind sehr komplex und benötigen sehr viele Ressourcen, die ihrem Einsatz negativ gegenüberstehen. Der Bedarf an Speicherkapazität, um die Routing Tabellen zu verwalten, übersteigt die gegebenen Voraussetzungen. Erschwerend kommt hinzu, dass in der Zeit, in der die Informationen über den Verbindungsstatus anderer Router eingehen, sich dieser schon wieder geändert haben könnte. Der Algorithmus operiert also auf veralteten Informationen was sich aufgrund der Frequenz der Dynamik auch nicht ändern wird. Ein Link State Algorithmus würde also unter seiner Leistung bleiben.

Mit dem Problem der Netzwerkdynamik haben auch Distance Vector Algorithmen zu kämpfen. Sie konvergieren langsamer als die Link State Pendanten, weswegen ihre Leistung unter den Erwartungen bleiben wird. Auch sie benötigen für die Verwaltung der Informationen Routing Tabellen, was sich nicht mit der Anforderung an geringen Speicherverbrauch deckt. Der Austausch von Teilen der Routing Tabellen belastet ausserdem die zur Verfügung stehende Bandbreite sehr stark. Für FAN wird daher geschlussfolgert, dass ein Austausch von routerbetreffenden Informationen zwischen Routern vermieden werden sollte.

Als Resultat der Analyse stehen folgende Anforderungen, an ein Routing Algorithmus für FAN:

- dynamisch, d.h. keine vorgegebenen Routen,
- Routingentscheidung in jedem zwischen Quelle und Ziel liegenden Router treffen, da ein aktuelles Gesamtbild des Netzwerkes nicht erhalten werden kann,
- Keine Routing Tabellen anlegen, da Speicherkapazität und Rechenleistung für Verwaltung stark begrenzt sind,
- möglichst kein Austausch von Router relevanten Daten, da die Bandbreite sehr klein ist,
- einfacher Algorithmus, mit simpler Implementierung,
- geringer Ressourcenbedarf des Algorithmus.

Weiterhin soll der Algorithmus auch die in Abschnitt 2.2 aufgestellten Ziele möglichst sehr gut erfüllen.

2.4.2 Eingesetzte Routing Algorithmen

Um die oben genannten Anforderungen zu erfüllen, werden folgende drei Routing Algorithmen vorgestellt:

- Flooding Routing
- Hot-Potato Routing
- Simple Hot-Potato Routing

Bei allen Algorithmen handelt es sich um dynamische Algorithmen, die keine Routing Tabellen aufbauen, keine Routerinformationen austauschen und einfach zu implementieren sind. Die Entscheidung über den Weg eines Datenpaketes wird dabei von jedem Algorithmus vor Ort, d.h. in jedem Router, und unabhängig von anderen Routern getroffen.

Flooding Routing

Dies ist eine sehr einfache Variante des Routing, auch als Fluten bezeichnet. Datenpakete, die über einen Link an dem Router ankommen, werden an alle verbleibenden Links des Router wieder gesendet. Hat ein Router nur einen Link und markiert damit einen Endpunkt im Netzwerk, dann wird ein darüber ankommendes Paket verworfen. Der Vorteil des Fluten besteht darin, dass alle möglichen Wege gegangen werden, um das Ziel zu erreichen, weswegen es sehr gut für dynamische Netzwerke mit unzuverlässigen Verbindungen geeignet ist. Der offensichtliche Nachteil ist aber die Verschwendung der Bandbreite durch viele duplizierte Nachrichten und die daraus resultierende Mehrarbeit (Overhead) für alle Router, um die Paketflut wieder einzudämmen.

Hot-Potato Routing

Einen anderen Ansatz verfolgt Hot-Potato Routing [Baran, 1964]. Es überflutet das Netz nicht, sondern jeder Router entscheidet anhand der Auslastung seiner Links, über welchen das Datenpaket weitergeleitet wird. Ausgenommen hierbei ist wie bei Flooding der Link, über den das Paket den Router erreichte. Dieser Algorithmus nutzt die zur Verfügungen stehenden Ressourcen sehr gut aus.

Simple Hot-Potato

Dieser Algorithmus ist eine vereinfachte Version des Hot-Potato Routing. Er funktioniert in gleicher Weise, allerdings geschieht die Wahl des Wegewahl nicht anhand der Auslastung aller Links, sondern über einen Zufallsgenerator. Ziel ist es dabei eine gleichmässige Verteilung der Datenpakete zu erreichen, indem sie durch den Zufallsgenerator gleichmässig auf alle Links verteilt werden, aber dabei das Netz nicht zu fluten. Offensichtlicher Nachteil dieses Algorithmus ist, dass Pakete auch über völlig überlastete Links zu senden versucht werden.

Allen Algorithmen ist gemein, dass sie keine Rücksicht auf den Verbindungsstatus der Links nehmen, über die sie gesendet werden. Auch besitzen sie kein Wissen über das aktuelle Bild des Netzwerkes. Die aufgestellten Anforderungen an ein Routing Algorithmus machen sie aber geeignet. Wie gut sie die in Abschnitt 2.2 aufgestellten Ziele erreichen, wird in Simulationen gezeigt.

3 Simulation und Implementierung

Um die Routing Algorithmen in FAN zu testen, wurde das Mittel der Simulation gewählt. Damit ist es möglich, Aussagen über die Leistungsfähigkeit eines Algorithmus in einer Vielzahl von Netzwerksituationen zu treffen. Durch den deterministischen Ablauf in einem Simulator kann man ausserdem verschiedene Routing Algorithmen hinsichtlich gewählter Metriken vergleichen. Unerwartetes Verhalten eines Protokolls kann ebenso erklärt werden, wie auch Ursachen für Fehler im Design oder in der Implementierung eines Routing Protokolls gefunden werden können.

In diesem Abschnitt wird eine Architektur für Simulationen des FAN vorgestellt. Im Anschluss daran werden verschiedene Implementierungsdetails besprochen.

3.1 FAN Simulations Architektur

Die FAN Simulations Architektur beschreibt das Design für eine Testumgebung, in der das Verhalten des Fabric-Area-Network simuliert wird. In dieser Umgebung können Routing Protokolle entworfen, ausgetestet und evaluiert werden. Das Ziel ist es eine experimentelle Vorstellung zu bekommen, welches Protokoll am besten für FAN geeignet ist.

3.1.1 Architektur - Überblick

Die Architektur wird geprägt von zwei Modulen. Das ist zum einen der Simulator und zum anderen ein Datenverarbeitungsmodul (siehe Abbildung 2).

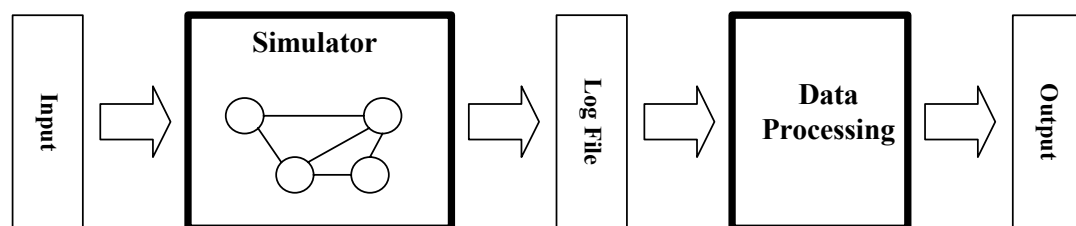


Abbildung 2 FAN Simulations Architektur

Beide Module sind können voneinander unabhängig implementiert werden. Sie besitzen die gemeinsame Schnittstelle über das Log File. Dieses enthält die kritische Masse an Information, die das simulierte System zu jedem Zeitpunkt vollständig beschreibt. Das Log File wird vom Simulator aus dem Input erzeugt. Der Input beschreibt das Verhalten von allen Komponenten (Netzwerk, Stationen, Buffer, Links), aus denen eine FAN Simulation aufgebaut ist. Nachdem der Simulator mittels der Inputs das Log File erstellt hat, erzeugt das Datenverarbeitungsmodul daraus den gewünschten Output. Dieser kann von einfachen Statistiken, verschiedenen Graphen für verschiedene Aspekte, die beleuchtet werden sollen, bis hin zur Aufbereitung des Simulationslaufes als graphische Animation reichen. Da das Log File dabei nicht zerstört wird, kann jederzeit ein neues Datenverarbeitungsmodul auf die Informationen zurückgreifen und seine Sicht der Daten erzeugen.

3.1.2 Simulator und seine Komponenten

Das Simulatormodul arbeitet auf Komponenten, die das FAN beschreiben. Folgende Anforderungen werden an dieses Modul gestellt:

- Der Simulator ist deterministisch. Diese Eigenschaft garantiert, dass jede Simulation reproduzierbar ist und bei wiederholter Ausführung immer wieder zu den gleichen Ergebnissen führt. Damit ist es möglich Anomalien oder ein spezielles Verhalten des Netzwerkes besser zu analysieren.
- Der Simulator ist ein diskreter Ereignis-Simulator. Es ist kein Echtzeit Simulator, sondern der Lauf wird in diskrete Ereignisse unterteilt.

Das ermöglicht, einen exakten Einzelschritt-Ablauf, der aufzeigt, was in jedem Zeitpunkt passierte.

- Komponenten, aus denen eine Simulation aufgebaut ist, sind konfigurierbar. Das Verhalten der Komponenten zu jedem Zeitpunkt, kann ohne Veränderung des Simulators beschrieben werden. Der Simulator selbst ist damit nur ein Kernel.

Während der Analyse des Aufbaus und der Funktionsweise des Fabric-Area-Network wurden vier grundlegende Komponenten mit ihrem Verhalten identifiziert, aus denen jede Simulation aufgebaut ist. In Tabelle 1 sind die Komponenten und deren Verhalten zusammengefasst.

Komponente	Verhalten	Charakter
Station (S)	Router-Verhalten Produzent-Verhalten Konsument-Verhalten	Dynamisch Dynamisch Dynamisch
Buffer (B)	Grösse Policy	Statisch Dynamisch
Link (L)	Bandbreite Latenz halb/voll-duplex	Statisch Statisch Dynamisch
Netzwerk (N)	Grösse (Anzahl der Stationen) Anzahl der Links einer Station Konnektions-Matrix (symmetrisch)	Statisch Statisch Dynamisch

Tabelle 1 Simulator-Komponenten und Verhalten

Unterschieden wird innerhalb des Verhaltens von Komponenten zwischen dynamischem und statischem Verhalten. Statisches Verhalten wird zu Beginn der Simulation durch einen Parameter festgelegt und ändert sich während des gesamten Ablaufes nicht. Ist ein Verhalten einer Komponente dynamisch, so kann es sich während des Simulationslaufes ändern. Dynamisches Verhalten wird durch eine Funktion mit der aktuellen Simulationszeit als Parameter beschrieben.

Die Stationen entsprechen zum grossen Teil der Treiberhardware, an denen z.B. Sensoren als Produzenten angeschlossen sind. Über dieser Hardware werden auch die Spulen als Verbindungen (Link) betrieben. Jeder Link ist dabei mit einem Buffer ausgestattet.

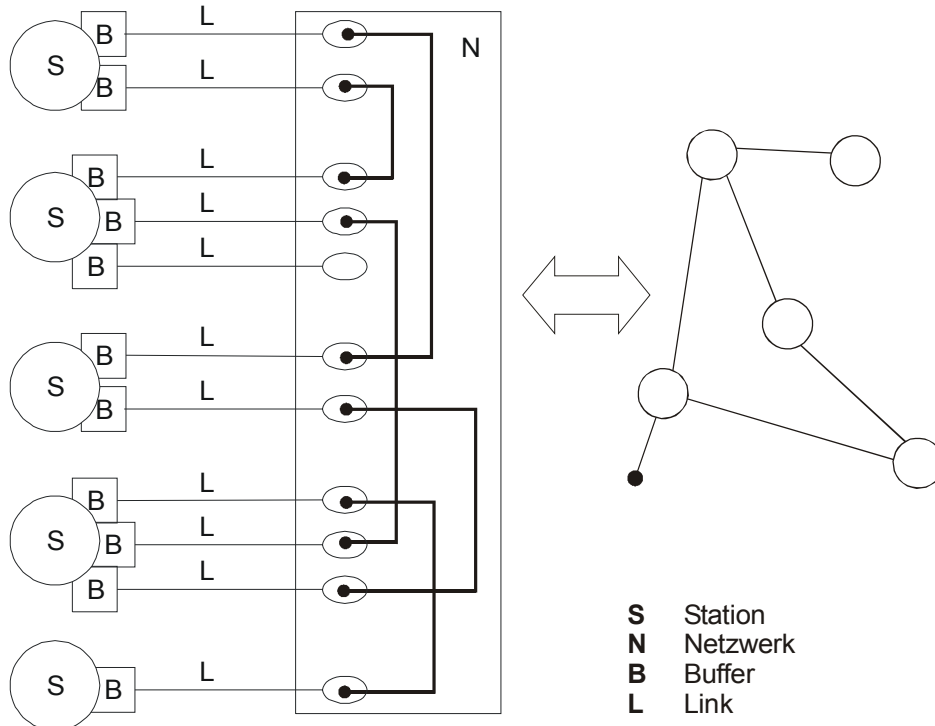


Abbildung 3 Simulator-Komponen und Netzwerk

Abbildung 3 zeigt die Zuordnung der Simulator-Komponenten auf ein Netzwerktopologie. Eine zentrale Stelle nimmt dabei die Konnektionsmatrix ein. Sie beschreibt die Verbindungen zwischen den Komponenten, die sogenannte Topologie. Dieses Verhalten ist dynamisch, wird also durch eine Funktion repräsentiert, wodurch das Netzwerk in seiner Topologie dynamisch wird. Dies ist genau einer der Hauptcharakteristika des FAN, die der Simulator damit erfüllt.

Die gesamte FAN Simulation Architektur ist erweiterbar. Der Simulator und das Datenverarbeitungsmodul sind zwei getrennte Module, die unabhängig voneinander implementiert werden können. Damit kann der Simulator als Kernel und die Daten des Log Files immer wieder verwendet werden, um aus verschiedenen Arten der Eingaben (GUI, Textdatei, ...) verschiedene Arten der Ausgaben (graphische Darstellung, statistische Daten, ...) zu erzeugen. Dies führt zu einem erweiterten Bild der FAN Simulations Architektur, welche in Abbildung 4 dargestellt wird.

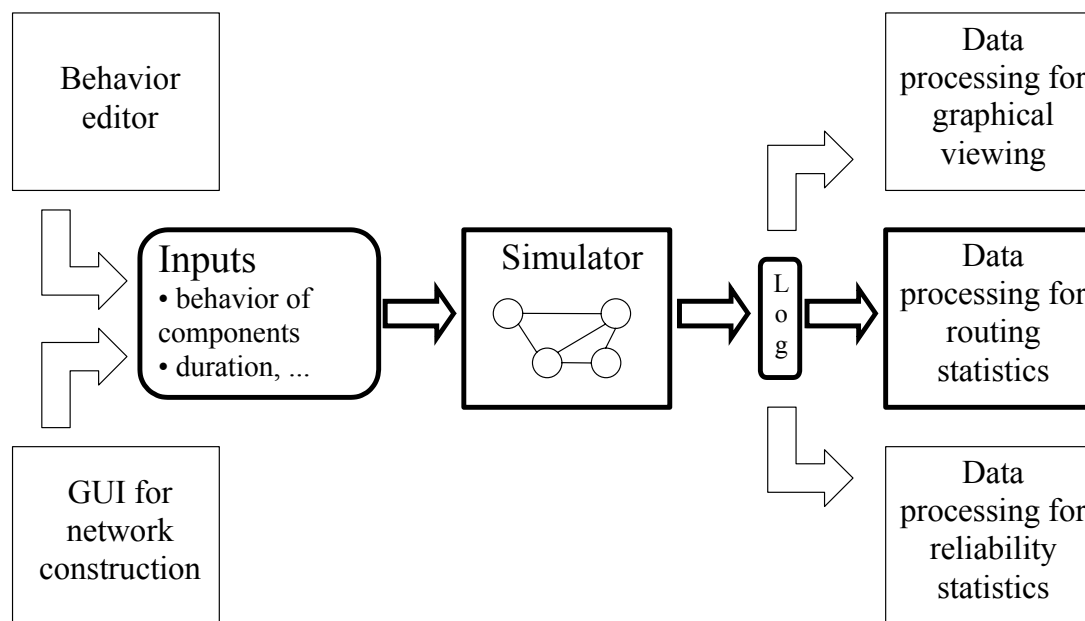


Abbildung 4 Erweiterte FAN Simulations Architektur

3.2 Implementierung der FAN Simulations Architektur

Um die Implementierung der Architektur auf die Programmierung der Komponenten und ihres Verhalten (siehe Tabelle 1) zu beschränken, wurde nach einem System gesucht, welches deterministische diskrete Ereignis-Abläufe steuert. Damit würden zwei wesentliche Anforderungen an den Simulator erfüllt werden. Zwei dieser Umgebungen, die dieses leisten, wurden ausgewählt und miteinander verglichen. Die erste Umgebung ist die bekannte ns-2 [ns-2, 2001a] Umgebung, die zweite ist die Ptolemy II Umgebung für heterogenes Design und Modellierung [Davis et al., 2001].

3.2.1 Vergleich der Umgebungen ns-2 und Ptolemy II

Beiden Umgebungen unterstützen die Implementierung von Abläufen auf der Basis von diskreten Ereignissen. Dabei handeln sie deterministisch und sind sehr gut getestet. Die umfangreiche Dokumentation, die beide bieten, hilft ebenfalls bei Implementierung der vorgestellten Architektur.

Ns-2 ist ein diskreter Ereignis Simulator, der für die Forschung an Netzwerken entwickelt wurde. Er unterstützt umfassend die Simulation von TCP Routing, Multicast Protokollen, ad-hoc Routing Protokollen und vieles mehr. Weiterhin existieren eine Reihe von Tools, die für diesen Netzwerksimulator entwickelt wurden, so zum Beispiel ein Topologie-Generator [Calvert, Doar & Zegura, 1997], oder auch ein Traffic-Generator [Henderson & Sahouria]. Die Entwicklergemeinschaft ist sehr gross und der Simulator wird von verschiedenen

Forschungsgruppen eingesetzt. Einen Auszug bietet [ns-2 2001b]. Für die Implementierung von Routing Protokollen muss man sich sehr tief in diesen komplexen Simulator einarbeiten, da solche Protokolle mit einer Vielzahl von ns2-Modulen interagieren muss. Es wird in C++ und Tcl programmiert, was das Verständnis des komplexen Zusammenspiels von Modulen nicht vereinfacht. Der Umstand, dass FAN kein „traditionelles“ Netzwerk ist, ermöglicht ein Verhalten der Implementierung, das mit dem Verhalten existierender Netzwerke nicht vergleichbar ist. Insbesondere können unbeabsichtigte Seiteneffekte verursacht werden.

Im Gegensatz zu ns-2 ist Ptolemy II kein reiner Netzwerksimulator, sondern eine Menge von Java-Paketen für heterogene, nebenläufige Modellierung und Design. Er ist ein Programmier-Rahmenwerk für eine Vielzahl von Modellen, wie diskrete Ereignis-Modelle, aber auch Datenfluss-Modelle, Prozess-Netzwerke u.v.a. Ptolemy II ist vollständig in Java implementiert und bietet damit einen einheitlichen Weg für die Implementierung. Es basiert auf einem komponentenbasierten Ansatz, um ein Model zu bauen. Die Möglichkeit des dynamischen Klassenladens macht es sehr einfach, die dynamischen Verhalten der in Tabelle 1 identifizierten Komponenten mit ihrem Verhalten zu konfigurieren. Ptolemy II liefert mit Vergil eine graphische Benutzerschnittstelle, mit der man die Komponenten in einfacher Art und Weise miteinander verbinden und konfigurieren kann. Die mit Vergil konstruierten Modelle werden dabei in MoML, einem XML Schema, gespeichert. Weiterhin stehen schon eine Vielzahl von Komponenten mit graphischen Fähigkeiten zur Verfügung, die als ein graphischer on-the-fly Analytiker dienen. Ein grosser Nachteil dieser Umgebung ist Ressourcennutzung durch den Gebrauch von Java. Es ist zu erwarten, dass Simulationsmodelle im allgemeinen langsamer laufen als vergleichbare Implementierungen in C++.

Was den Aufwand der Implementierung betrifft, so sind beide System in etwa gleich. Für den ns-2 muss vor der Implementierung der FAN Simulations Architektur der Aufwand betrieben werden, um seine komplexe Struktur zu verstehen. Ptolemy II stellt für die Architektur lediglich die Steuerung des diskreten Ereignis-Ablaufes zur Verfügung und überlässt es dem Entwickler, wie er die Komponenten und deren Kommunikation untereinander gestaltet. Aufgrund dieser freien Gestaltungsmöglichkeiten, die die Voraussetzungen sind, um möglichst flexibel zu sein, und der Möglichkeit des dynamischen Klassenladens, wurde Ptolemy II als Plattform für die Implementierung der FAN Simulations Architektur gewählt.

3.2.2 Implementierungsdetails

Der erste Schritt in der Implementierungsphase war die Entwicklung der Komponenten aus Tabelle 1. Die Station wurde in den drei Ausprägungen (Producer, Consumer, Router) entwickelt, die ihrem möglichen Verhalten entsprechen. Der Producer kennt zwei Argumente. Eine Producer-Funktion als erstes Argument, die die Token-Funktion, welche die Datenpakete produziert, triggert. Der Consumer wird durch die Consumer-Funktion bestimmt, die die Anzahl der in einer bestimmten Zeit zu konsumierenden Datenpakete beschreibt. Der komplexeste Teil wird vom Router übernommen. Sein Verhalten wird nur durch eine Funktion, der Routing-Funktion, beschrieben, die ankommende Datenpakete auf mögliche angeschlossene Ausgabelinks gemäss dem implementierten Algorithmus verteilen muss. Implementiert wurden die in Abschnitt 2.4.2 vorgestellten Routing Algorithmen, Flooding, Hot-Potato und Simple Hot-Potato.

Der Buffer kann durch zwei Argumente konfiguriert werden. Zum ersten ist das die Grösse als statischer Parameter und zum zweiten die Policy-Funktion, die die auch das Speichern der Pakete im Buffer übernimmt. Bisher wurde nur die FIFO Policy implementiert. Über die drei Argumente Bandbreite, Latenz und Resolver-Funktion wird der Link konfiguriert. Die ersten beiden sind dabei als statische Parameter implementiert. Die Resolver-Funktion legt fest, wer einem Sender/Empfänger-System senden und empfangen darf. Damit wird der Halb- bzw. Vollduplex Betrieb eines Links implementiert. In einer Simulation verfügt jeder Link über einen Sende- und einen Empfangsbuffer. Bisher wurde nur der Halbduplex-Betrieb eines Links implementiert.

Die Netzwerk-Komponente benötigt als Argumente die Anzahl der im Netzwerk eingesetzten Stationen und die Anzahl der Links pro Station. Daraus wird dann die symmetrische Konnektionsmatrix auf gebaut. Alle Stationen im Netzwerk haben die gleiche Anzahl von

Links. Soll eine Station mit weniger Links modelliert werden, so werden die überschüssigen Links durch die Konnektionsmatrix nicht miteinander verbunden. Das dritte Argument der Netzwerkkomponente ist die Netzwerk-Funktion, die den Verbindungsstatus aller Links zu jedem Zeitpunkt der Simulation beschreibt.

Da Ptolemy II einen komponentenbasierten Ansatz verfolgt, können Komponenten auch zu komplexeren Komponenten zusammengesetzt werden. So wurden Link, Buffer und Router (Ausprägung der Station-Komponente) zu einem LinkRouter zusammengesetzt. Der Simulator ist auch für die Erstellung der Log Files verantwortlich (siehe Abbildung 2). Für diesen Zweck wurden Beobachter (Observer) eingeführt. Diese werden mit dem Namen der zu beobachteten Komponente und einer Schreib-Funktion, die für die Erstellung und das Format der Log Files zuständig ist, konfiguriert. Alle Komponenten werden durch einen Zeitgeber (Timer) getriggert, der sie miteinander verbindet. Zwischen den Komponenten werden Daten in Form von Token ausgetauscht. Auch die generierten Datepakete werden in solche Token eingebettet. Die Abbildung 5 zeigt ein in Vergil konstruiertes Modell einer Simulation aus zwei Stationen, die über einen Link miteinander verbunden sind.

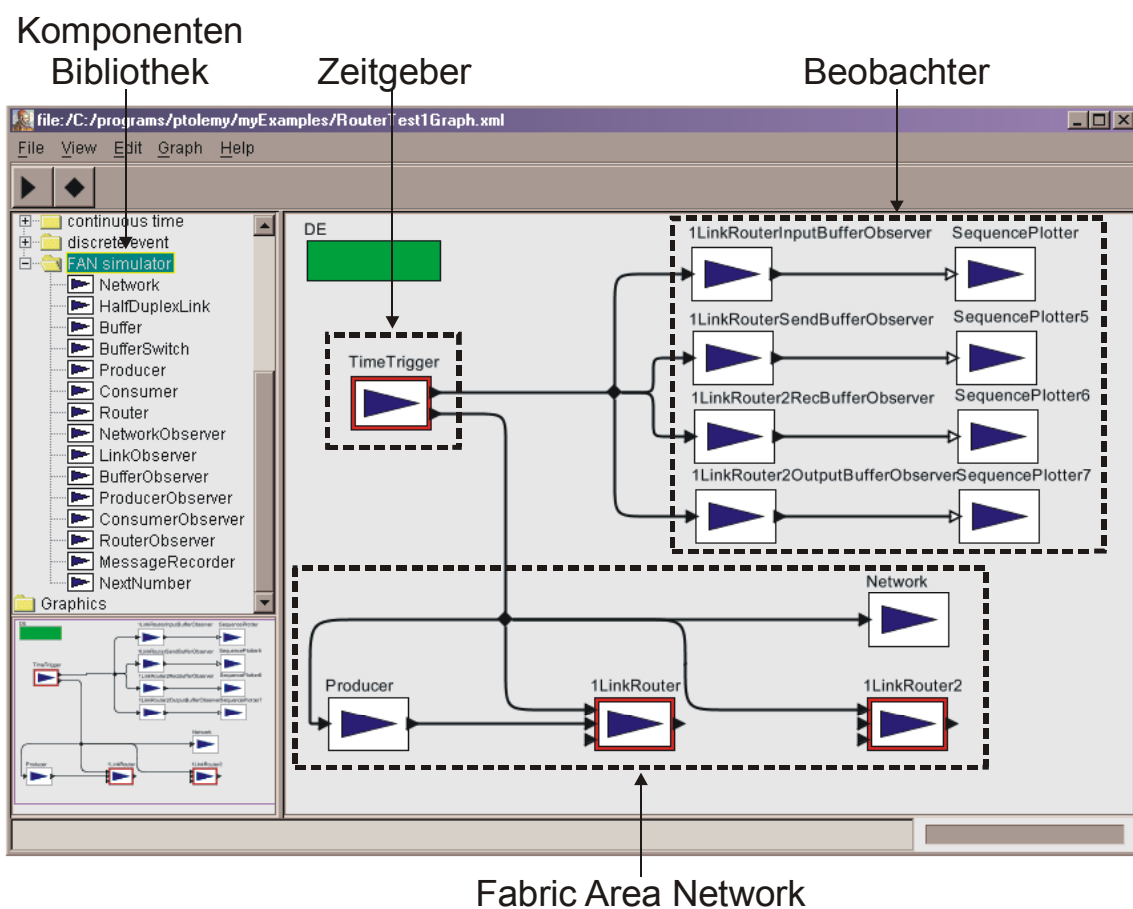


Abbildung 5 FAN Simulationsmodell in Vergil

Verschiedene Buffer werden in dem Modell von entsprechenden Beobachtern überwacht. Der Trigger taktet jeden Bestandteil des Modell. Verbindungen wie zwischen Producer und 1LinkRouter, transportieren Datenpakete in Form von Token. Zwischen 1LinkRouter und 1LinkRouter2 ist eine solche Verbindung nicht zu sehen, denn diese werden von der Netzwerkkomponente (Network) durch die Konnektionmatrix verwaltet. Das Datenverarbeitungsmodul kann wie in Abschnitt 3.1 dargestellt, unabhängig vom Simulator Kernel implementiert werden. Je nach gewünschten Ergebnissen wurde dieses Modul in verschiedenen Skriptsprachen, wie Perl und PHP programmiert.

3.2.3 Einschränkungen und zukünftige Arbeiten

Die derzeitige Implementierung des Simulator Moduls der FAN Simulations Architektur besitzt folgende Einschränkungen:

- Buffer können bisher nur die FIFO Policy nutzen,
- Links können nur im Halb-Duplex-Modus betrieben werden,
- alle Links arbeiten parallel, d.h. wenn der Sendbuffer des Links Datenpakete enthält, dann werden die auch gesendet,
- Zuordnung von unterschiedlichen Zeitintervallen, die Komponenten für ihre Aufgaben brauchen, ist bei sehr kleinen Zeitschritten (< 1 ms) nicht möglich,
- pro Zeitschritt verarbeitet die Routing Funktion ein Paket einer angeschlossenen Quelle und ein Paket, welches von anderen Routern gekommen ist.

Andere Buffer Policies und Links auf voll-duplex Betrieb umstellen bedarf der Entwicklung neuer Funktionen, die das dynamische Verhalten von Buffer und Link beschreiben. Die Arbeitsweise des Simulator Kernel bleibt damit unangetastet. Für die weitere Zukunft ist eine genauere Steuerung der Links geplant. Dies ist sinnvoll im Hinblick auf eine genauere Simulation des Energiehaushalt der Treiberhardware. Ein allgemeines Ziel für Zukunft ist die Steigerung der Ausführungsgeschwindigkeit des gesamten Simulator.

4 Routing - Simulationsergebnisse

Die in Tabelle 1 aufgezählten Komponenten weisen eine Vielzahl von Verhalten auf. Im Rahmen dieser Studienarbeit wird ein Bottom-Up-Ansatz benutzt, um die Einwirkungen dieser Parameter auf Routing Entscheidungen und damit auf die Leistungsfähigkeit der verwendeten Routing Protokolle zu untersuchen. Der Ansatz besteht darin, alle diese Verhalten konstant zu halten und nur ein Parameter zu variieren. Sich ergebende Resultate können auf den Einfluss dieses Verhaltens zurückgeführt werden. Dies führt zu einer Vielzahl von Möglichkeiten, die es zu untersuchen gilt. Im Rahmen einer Studienarbeit konnte nur ein Teil der Möglichkeiten untersucht werden. Deshalb wurde zwei wichtige Grössen ausgewählt: Zum ersten wurde der Ressourcenbedarf der Routingprotokolle untersucht und zum zweiten die Netzwerkdynamik. Es werden dabei keine grossen Netzwerke mit komplizierten Topologien als Modell für die Simulation herangezogen, sondern sehr einfache Modelle. Als Routing Protokolle kommen Flooding, Hot-Potato und Simple Hot-Potato in der Form, wie sie in Abschnitt 2.4.2 besprochen wurden, zum Einsatz.

Ziel ist es, herauszufinden welchen Einfluss die untersuchten Grössen auf das Routing Verfahren haben, wo Schwachpunkte liegen, aber auch wo Potenzial für Verbesserungen liegt. Diese Daten stellen dann die Basis für Schlussfolgerungen in grösseren Fabric-Area-Networks dar. Dabei ist es das Ziel durch Wahl des passenden Routing Protokolls, die Leistungsfähigkeit und Ressourcennutzung der einzelnen Station und damit auch des gesamten Netzwerkes zu optimieren.

4.1 Untersuchung des Ressourcenbedarfs

Im FAN stehen nur wenige Ressourcen zur Verfügung. Daher ist es wichtig, den Bedarf dieser Ressourcen zu kennen. In diesem Abschnitt wird die Frage behandelt, welche Ressourcen in welchem Umfang gebraucht werden, damit ein Routing Protokoll Pakete so auf einzelne Links verteilen kann, so dass möglichst wenig Pakete verworfen werden. Die Betrachtung der Ressourcen Energie, Speicherplatz für das Protokoll und aufzuwendende Rechenleistung wird hier vernachlässigt, da keine technischen Daten oder Schätzungen vorhanden sind.

In jeder hier vorgestellten Simulation hat ein Link einen Sendebuffer und einen Empfangsbuffer. Empfangsbuffer werden nur so schnell gefüllt, wie der ein Link mit seiner Bandbreite und Latenzzeit schnell ist. Um ein 18 byte Paket über einen Link mit einer Bandbreite von 1000 bits/s und einer Latenzzeit von 100ms zu transportieren, werden etwa 240 ms benötigt. Ein Router kann aber innerhalb von 10 ms ein Paket aus einem Empfangsbuffer entnehmen. Bei mehreren Links pro Station (in den durchgeführten Simulationen waren es maximal drei), werden $(\text{Zahl der Links pro Station} \times 10)$ ms benötigt, bis der Router aus jedem Empfangsbuffer ein Paket entnommen hat. Das liegt immer noch unter der Latenzzeit eines Links. Aus diesem Grund werden die Empfangsbuffer in der Betrachtung vernachlässigt.

Die Untersuchung des Einflusses von Sendebuffer-Grössen führt zu der Frage, wie schnell, welche Datenmengen im Buffer abgelegt werden und wie schnell sie dort wieder herausgenommen werden. Routing Algorithmen stehen dabei in einer Schlüsselposition, denn sie entscheiden, wieviele Datenpakete in den Sendebuffer gelegt werden. Die Frequenz eingehender Datenpakete hängt im einfachsten Fall nur vom Produzenten ab. Die Herausnahme der Pakete aus dem Buffer hängt von den Linkeigenschaften und damit von Bandbreite und Latenzzeit ab. Da die Latenzzeit durch technische Eigenschaften fest ist, wird die Bandbreite zur wichtigsten Ressource der Links. Wichtige Ressourcen, die in diesem Abschnitt untersucht werden, sind Sendebuffer-Grössen und Bandbreiten.

4.1.1 Simulation - Topologie und Parameter

Um die verschiedenen Routing Protokolle zu untersuchen, stehen mehrere Sendebuffer zur Verfügung, auf die die Datenpakete des Produzenten gemäss dem Routing Algorithmus verteilt werden. Als Topologie wurden zwei Stationen gewählt, wobei die erste mit einem Datenpaket-Produzenten verbunden ist und versucht mittels Routing Protokollen diese

Pakete an die zweite Station zu übermitteln. Die zweite Station ist nur eine Empfangsstation. Dieser Aufbau ist Abbildung 6 wiedergegeben.

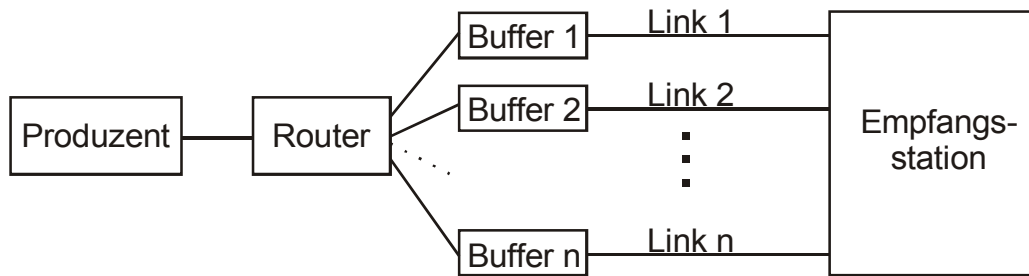


Abbildung 6 Topologie für Ressourcenuntersuchung

Für $n = 1,2,3$ wurden Simulationen durchgeführt. In jeder dieser Simulationen wurden Produzent, Routing Protokoll, Buffergrösse und Bandbreite der Links geändert. Für alle $n = 1,2,3$ wurde somit jede Kombination dieser Grössen simuliert. Im Anschluss daran wurde die Leistungsfähigkeit der Routing Protokolle anhand gewählter Metriken verglichen. Es wurden drei unterschiedliche Produzenten ausgewählt, deren Eigenschaften in Tabelle 2 angegeben sind.

4.1.1.1 Produzent	4.1.1.2 Produktionsrate
Producer 1	1 Datenpaket der Grösse 18 bytes alle 240 ms
Producer 2	10 Datenpakete jeweils 18 bytes alle 2400 ms
Producer 3	1 Datenpaket der Grösse 18 bytes alle 240 ms <i>und</i> 10 Datenpakete jeweils 18 bytes alle 2400 ms

Tabelle 2 Produzenten und Produktionsraten

Als Routing Protokolle wurden Flooding, Hot-Potato und Simple Hot-Potato eingesetzt. Alle verwendeten Sendebuffer sind nach der FIFO Policy organisiert. Ihre Grösse variiert von 0 bytes bis 360 bytes in 18 bytes Schritten, d.h. die Buffer können 0 bis 20 Datenpakete gleichzeitig halten. Die Latenzzeit aller Links ist unveränderlich 100 ms, wobei die Bandbreite von 100 bits/s bis 3000 bits/s (in 300 bits/s Schritten) variieren kann. Die Netzwerkfunktion beschreibt ein statisches Netzwerk. Die Simulationszeit betrug pro Parameterkonfiguration 240s, um mindestens 1000 Datenpakete zu erzeugen.

4.1.2 Simulation - Ergebnisse und Auswertung

Um die Leistungsfähigkeit der Routing Protokolle zu beurteilen, wurde als Metrik die Verlustrate der Datenpakete herangezogen. Paketverlust tritt auf, wenn mehr Pakete in den Buffer gelegt werden als herausgenommen werden. Finden Pakete keinen Platz mehr im Buffer, so werden sie verworfen. Ein Routing Protokoll ist um so besser, je weniger Pakete im Vergleich zu anderen Protokollen verloren gehen. Diese Verlustrate ist gemäss den Parametern des vorherigen Abschnittes von Bandbreite, Buffergrösse, Anzahl der Buffer und verwendetem Produzenten abhängig. Die Abbildung 7 zeigt exemplarisch die Abhängigkeit der Verlustrate von Buffergrösse und Bandbreite für den Produzenten 'Producer 2' unter Verwendung von Flooding Routing in der Topologie $n = 2$, in der 2 Stationen über 2 Links miteinander verbunden sind.

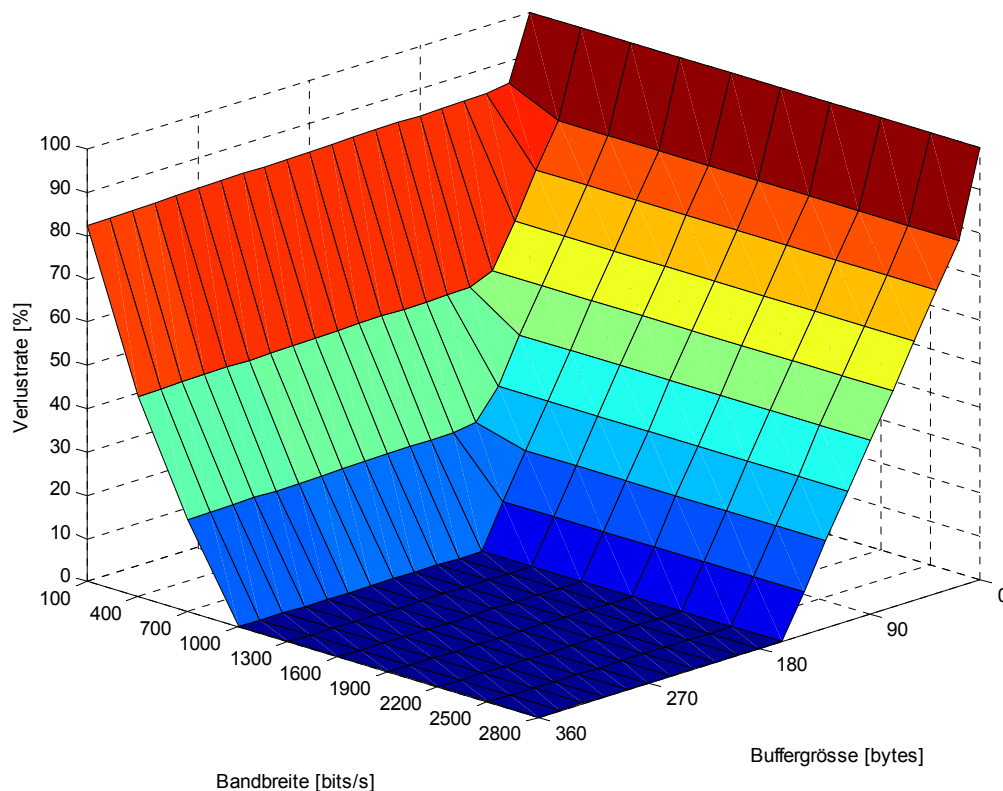


Abbildung 7 Verlustrate für Producer 2 unter Verwendung von Flooding für $n=2$

Aufgrund der vielen veränderlichen Grössen, wie Bandbreite, Buffergrösse, Art des Produzenten, Vernetzung, lassen sich mit Graphen wie in Abbildung 7 nur schwer vergleichende Aussagen über Routing Protokolle treffen. Deshalb wurden neue Metriken auf Basis der Verlustrate gesucht. In Abbildung 7 ist zu erkennen, dass ab einer Bandbreite von 1000 bits/s und einer Buffergrösse von 162 bytes kein weiterer Paketverlust auftritt. Genau diese Punkte, an denen kein Paketverlust auftritt werden in Abbildung 8 dargestellt. Sie bilden als Kurve eine Grenze zu einem ganzen Bereich über der Kurve, aufgespannt von Bandbreite und Buffergrösse, in dem die Verlustrate auf 0 % abgesunken ist.

Vorteil dieser Darstellung ist, dass man nun verschiedene Routing Protokolle in Abhängigkeit von Buffergrössen und Bandbreiten unter Verwendung eines Produzenten und in einer bestimmten Topologie untersuchen kann. Allerdings wird nur noch zwischen Paketverlust und Kein-Paketverlust unterschieden, es fehlen also genauere Aussagen zu Verlustraten.

Um Routing Protokolle produzentenübergreifend zu untersuchen, wird die Metrik Paketverlustfläche eingeführt. Sie beschreibt die Grösse der Fläche unter den Paketverlustgrenzen. Genaue Verlustraten werden nicht betrachtet, es wird lediglich festgestellt, dass in diesem Bereich Paketverlust auftritt. Werte für Buffergrössen und Bandbreiten werden durch diese Metrik zu einer Zahl substituiert, die den Ressourcenbedarf angibt, der mindestens notwendig ist, damit kein Paketverlust auftritt. Eine kleinere Verlustfläche bedeutet, dass das Routing Protokoll den produzierten Datenverkehr mit weniger Ressourcen handhaben kann. Das wird als eine höhere Leistungsfähigkeit des Protokolls angesehen. Die Paketverlustfläche gibt einen allgemeineren Überblick über den Ressourcenbedarf als Paketverlustgrenzen. Mit dieser Metrik kann man verschiedene Routing Protokolle abhängig vom Produzenten und Ressourcenbenutzung (Bandbreite und Buffergrösse) miteinander in einer Topologie vergleichen.

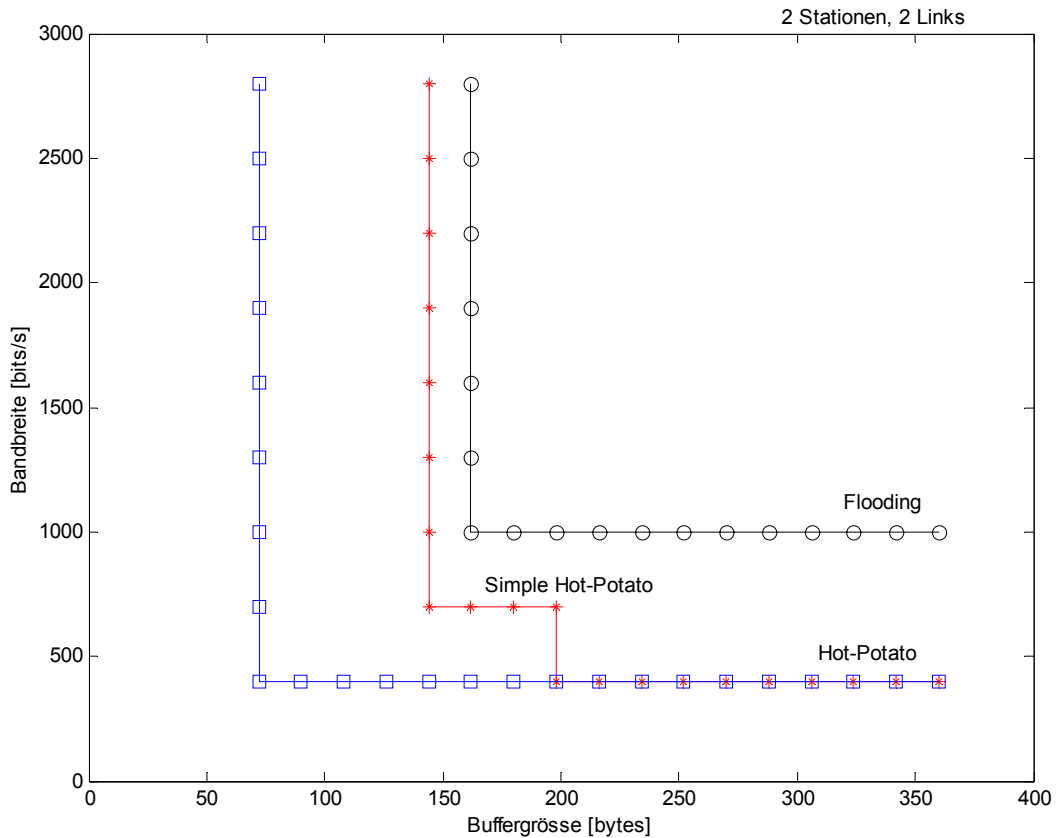


Abbildung 8 Paketverlustgrenzen aller Routing Protokolle für Producer 2 und n=2

Abbildung 9 zeigt für alle verwendeten Produzenten und Routing Protokolle die Paketverlustflächen für n=1 und n=2.

Man kann deutlich erkennen, dass Flooding in beiden Topologien und für alle Produzenten, die grösste Paketverlustfläche aufweist. Das heisst, dass es viel von den zur Verfügung stehenden Ressourcen (Bandbreite und Buffergrösse) braucht, bis keine Paketverluste mehr auftreten. Ursache dafür ist der Prozess des Flutens, durch den Pakete dupliziert werden, um sie auf alle Buffer zu verteilen. Es wird ausserdem deutlich, dass der Hot-Potato Algorithmus die Ressourcen am besten ausnutzen kann. Das liegt darin begründet, dass er zum ersten, im Gegensatz zu Flooding, die Pakete nicht dupliziert und zum anderen die Routing-Entscheidung vom Füllstand der Buffer abhängig macht. Sind mehrere Buffer vorhanden, so können Datenpakete sehr gut verteilt werden, und damit Ressourcen gespart werden. Darum ist eine weitere Ersparnis beim Übergang von n=2 auf n=3 noch möglich. Interessant ist, dass selbst mit einem zufälligen Verteilen der Pakete, wie das beim Einsatz von Simple Hot-Potato geschieht, eine deutliche Ressourceneinsparung im Vergleich zu Flooding für alle Produzenten erzielt werden kann.

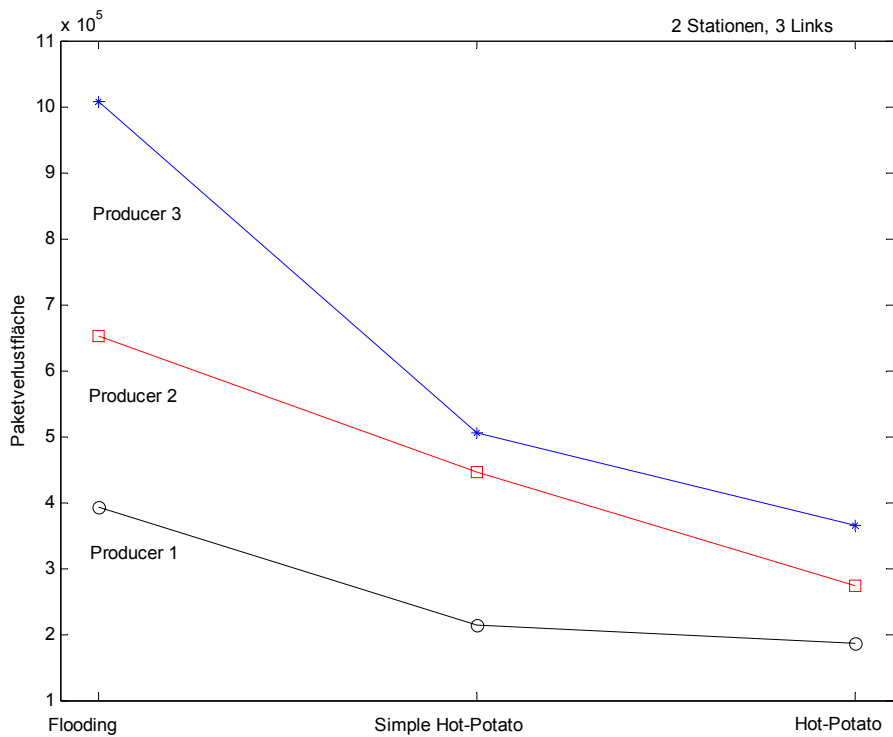
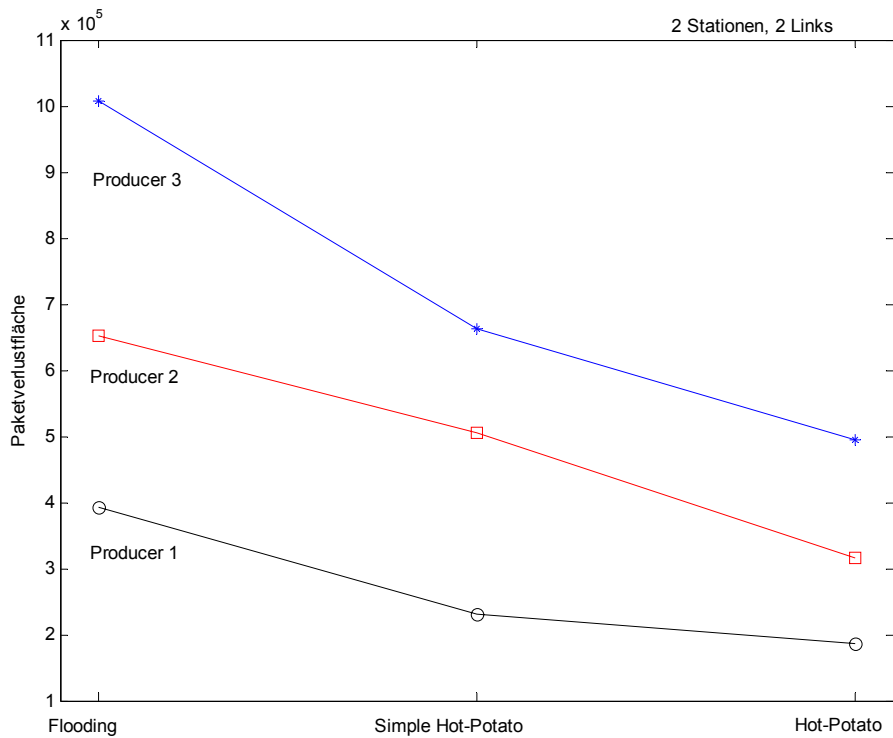


Abbildung 9 Paketverlustflächen für n=2 und n=3

Der Fall n=1 wird hier gesondert betrachtet. In diesem Fall sind zwei Stationen nur durch einen Link miteinander verbunden. Die Routing Algorithmen haben keine zweite Entscheidungsmöglichkeit, die sie nutzen könnten. Damit haben sie keinen Effekt auf die Verlustrate und die Paketverlustfläche ist direkt abhängig vom Produzenten. Abbildung 10 zeigt die Paketverlustfläche in Abhängigkeit von den verwendeten Produzenten für diesen

Fall. In dieser Abbildung kann man erkennen, dass die verschiedenen Produzenten unterschiedlich viel Ressourcen benötigen. Insbesondere der Produzent 'Producer 3' benötigt überdurchschnittlich viel Bandbreite und Kapazität in den Buffern.

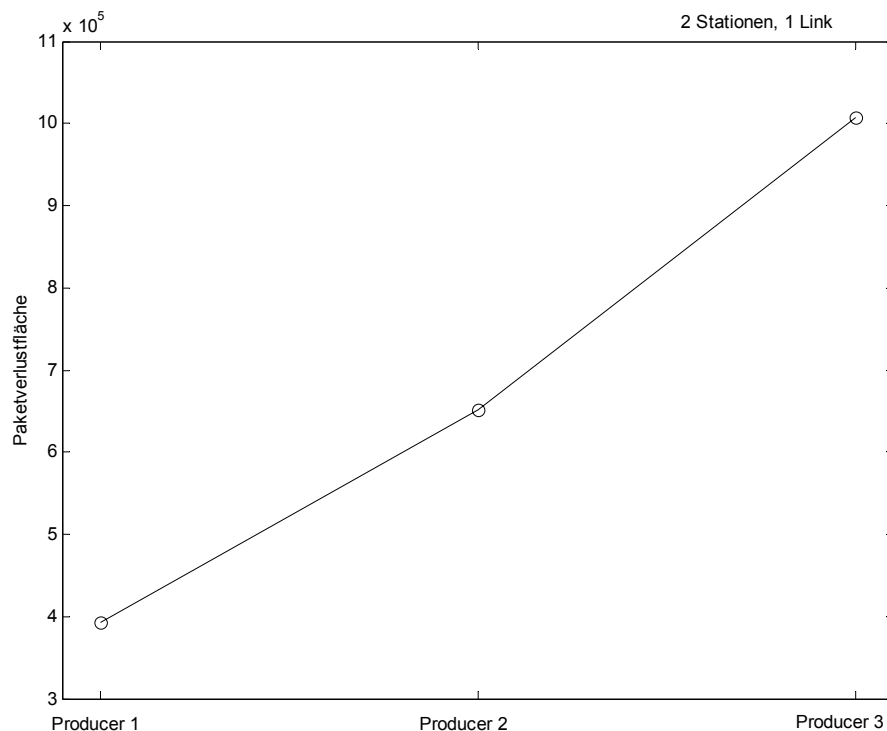


Abbildung 10 Paketverlustflächen für n=1

4.1.3 Simulation - Zusammenfassung

Die Untersuchung des Ressourcenbedarfs von Routing Protokollen konzentrierte sich auf Sendebuffergrößen und Bandbreiten der beteiligten Links. Dazu wurde die Metrik Paketverlustfläche eingeführt. Diese fasst Werte für Buffergrößen und Bandbreiten zu einem Ressourcenbedarf zusammen, der mindestens notwendig ist, um Pakete ohne Verluste über die vorhandenen Links zu transportieren. Dabei bedeutet eine kleinere Paketverlustfläche eines Protokolls einen geringeren Ressourcenbedarf. Alle Routing Protokolle brauchen weniger Ressourcen, wenn das erzeugte Datenaufkommen klein und sehr gleichmässig ist, wie das bei dem Produzent 'Producer 1' gegeben war.

Die Ergebnisse zeigen, dass das Hot-Potato Protokoll immer den geringsten Ressourcenbedarf hat. Die Metrik lässt aber keine Rückschlüsse auf die genauen Werte von Bandbreite und Buffergröße zu. Dazu müssen die Paketverlustgrenzen untersucht werden. Eine Analyse der Paketverlustgrenzen für alle Protokolle und alle Produzenten zeigte aber, dass Hot-Potato ebenfalls das Protokoll war, mit den geringsten Anforderungen an Bandbreite und Kapazität der Buffer.

4.2 Untersuchung der Netzwerkdynamik

Die FAN Verbindungen sind aufgrund der sich ständig ändernden Verhältnisse im Netzwerk sehr unzuverlässig. Daher besteht die Notwendigkeit, zu untersuchen, wie sich die Dynamik des Netzwerkes auf den Datentransport über solche Links auswirkt. Weiterhin wird untersucht, welchen Einfluss die Dynamik auf Entscheidungen der Routing Algorithmen hat, und welche Auswirkungen der Einsatz von Routing Protokollen auf den Datentransport hat. Es werden drei einfache Testfälle konstruiert, die ein typisches Verhalten des Netzwerkes simulieren. Wenn ein Link dabei ausfällt, geht alle Pakete auf diesem Link verloren; weiterhin können keine Pakete transportiert werden. Ausfallzeiten können zum Teil von Sendebuffer

kompensiert werden. Die eingesetzten Routing Protokolle können aber keine Informationen über den aktuellen Linkstatus verarbeiten. Sollte daher der Link über längere Zeit nicht zur Verfügung stehen, dann können die Sendebuffer auch überlaufen und damit Paketverlust verursachen. Damit wird die Verlustrate in Abhängigkeit von der Buffergrösse zu einer geeigneten Metrik. Andere Metriken wie die Verzögerungszeit oder auch die Weglänge (Hops), die ein Paket von der Quelle bis zum Ziel zurücklegt. Im Angesicht der hier verwendeten simplen Topologien ist die Verzögerungszeit eher unangemessen. Dagegen ist die Weglänge sehr gut als Metrik geeignet, zeigt sie doch welcher Weg zum Ziel bevorzugt wird. Ziel der Untersuchung ist es, Aussagen über die Qualität der Routing Protokolle im dynamischen Farbic-Area-Network zu treffen.

4.2.1 Simulation - Topologie und Parameter

Um das Verhalten in dynamischen Netzwerken zu untersuchen, wurden Beispieltopologien ausgewählt, die charakteristisch für FAN sind. Abbildung 11 zeigt wie die Stationen (S) in den verschiedenen Topologien miteinander verbunden sind.

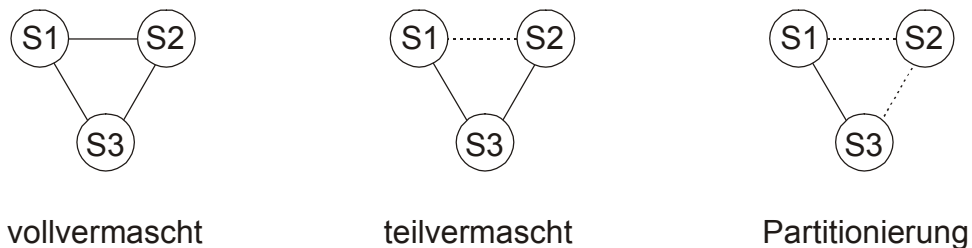


Abbildung 11 Topologien mit dynamischem Verhalten

In allen Topologien werden Datenpakete von einem Produzenten an S1 generiert und enthalten als Zieladresse die der Station S2. Datenpakete können direkt nach S2 geschickt werden, oder auch über auf dem Weg über Station S3 die Station S2 erreichen. Das vollvermaschte Netz stellt den Referenzfall dar, der ein statisches Netzwerk simuliert. Im teilvermaschten Netz unterliegt der Link S1-S2 einem dynamischen Verhalten, während der alternative Weg über Station S3 immer verfügbar ist. Im Partitionierungsfall führen beide Wege nach S2 über dynamische Links. Es kann der Fall auftreten, dass beide nicht verfügbar sind, d.h. dass die Station S2 vom restlichen Netzwerk getrennt ist. Ob ein Link zwischen zwei Stationen vorhanden ist wird durch Netzwerkfunktionen als Linkstatus in der Konnektivitätsmatrix (siehe Abschnitt 3.1.2) beschrieben. Tabelle 3 erläutert die Netzwerkfunktionen für den teilvermaschten Fall, während Tabelle 4 dies für den Partitionierungsfall durchführt.

4.2.1.1 Netzwerkfunktion	4.2.1.2 Verhalten
NetworkFunc1	ändert den Status des S1-S2 Link alle 2400 ms
NetworkFunc2	ändert den Status des S1-S2 Link alle 200 ms
NetworkFunc3	alle 100 ms mit der Status des S1-S2 Links geändert – mit einer Wahrscheinlichkeit von 50%

Tabelle 3 Netzwerkfunktionen für die teilvermaschte Topologie

Die ersten zwei Funktionen in Tabelle 3 ändern den S1-S2 Link sehr gleichförmig, wenn auch unterschiedlich schnell. Die dritte Funktion bringt dagegen das Zufallselement hinein.

4.2.1.3 Netzwerkfunktion	4.2.1.4 Verhalten
NetworkFunc1	ändert alle 2400 ms den Status des S1-S2 und des S3-S2 Links mit einer Wahrscheinlichkeit von 50%
NetworkFunc2	ändert alle 200 ms den Status des S1-S2 und des S3-S2 Links mit einer Wahrscheinlichkeit von 50%

NetworkFunc3	ändert alle 2400 ms den Status des S1-S2 Links und alle 200 ms den des S3-S2 Links mit einer Wahrscheinlichkeit von 50%
NetworkFunc4	ändert alle 200 ms den Status des S1-S2 Links und alle 2400 ms den des S3-S2 Links mit einer Wahrscheinlichkeit von 50%

Tabelle 4 Netzwerkfunktionen für die partitionierte Topologie

Die in Tabelle 4 vorgestellten Netzwerkfunktionen ändern den Status beider Links nach dem Zufallsprinzip. Dabei können alle Kombinationen vom vollvermaschten Netz bis hin zur Partitionierung von Station S2 auftreten. Über die übrigen Parameter der Komponenten in der Simulation gibt Tabelle 5 Aufschluss.

4.2.1.5 Komponente	4.2.1.6 Verhalten	4.2.1.7 Konfiguration
Station	Router-Verhalten Produzent-Verhalten	Flooding, Hot-Potato, simple Hot-Potato alle 240 ms wird ein 18 byte Datenpaket adressiert an S2 generiert
Buffer	Grösse Policy	18,...,360 byte (in 18 byte Schritten FIFO)
Link	Bandbreite Latenzzeit Halb/voll-duplex	1000 bits/s 100 ms halb-duplex
Netzwerk	Anzahl der Stationen Links pro Station Konnektionsmatrix	3 2 siehe Tabelle 3 und Tabelle 4

Tabelle 5 Simulationsparameter

Es kommt hinzu, dass für diese Untersuchung ein Timeout-Mechanismus in jedem Routing Protokoll Verwendung findet. Dieser sorgt dafür, dass nach 3s, ohne dass ein Paket aus dem Buffer entnommen wurde, der Bufferinhalt verworfen wird. Die Simulationszeit betrug 2400 s, wobei 10000 Datenpakete vom Produzenten erzeugt wurden.

4.2.2 Simulation - Ergebnisse und Auswertung

Um die Routing Protokolle miteinander vergleichen zu können und Ursachen für bestimmte Verhaltensweisen zu finden, wurden als Metriken die Verlustrate und die Weglänge (Hops) herangezogen. Insbesondere durch letztere kann man beurteilen, über welchen Weg die Datenpakete zur Station S2 gelangten. Buffer können bis zu einem bestimmten Maß Verlustraten begrenzen. Daher wird die Verlustrate in Abhängigkeit zu verschiedenen Sendebuffergrössen dargestellt.

Vollvermaschte Topologie

Die vollvermaschte Netztopologie (siehe Abbildung 11) wurde herangezogen, um einen Vergleich der Metriken in einem statischen Netzwerk zu haben. Die Parameter für die Simulation (siehe Tabelle 5) wurden gerade so gewählt, dass kein Paketverlust aufgetreten ist für alle verwendeten Buffergrössen. Hot-Potato hat immer den kürzesten Weg gewählt, somit also nur ein Hop zwischen S1 und S2 benötigt, während Simple Hot-Potato durchschnittlich 1.5 hops bis zum Zielstation S2 benötigte, da die Pakete durch den Zufallsgenerator gleichmässig auf beide Wege verteilt wurden.

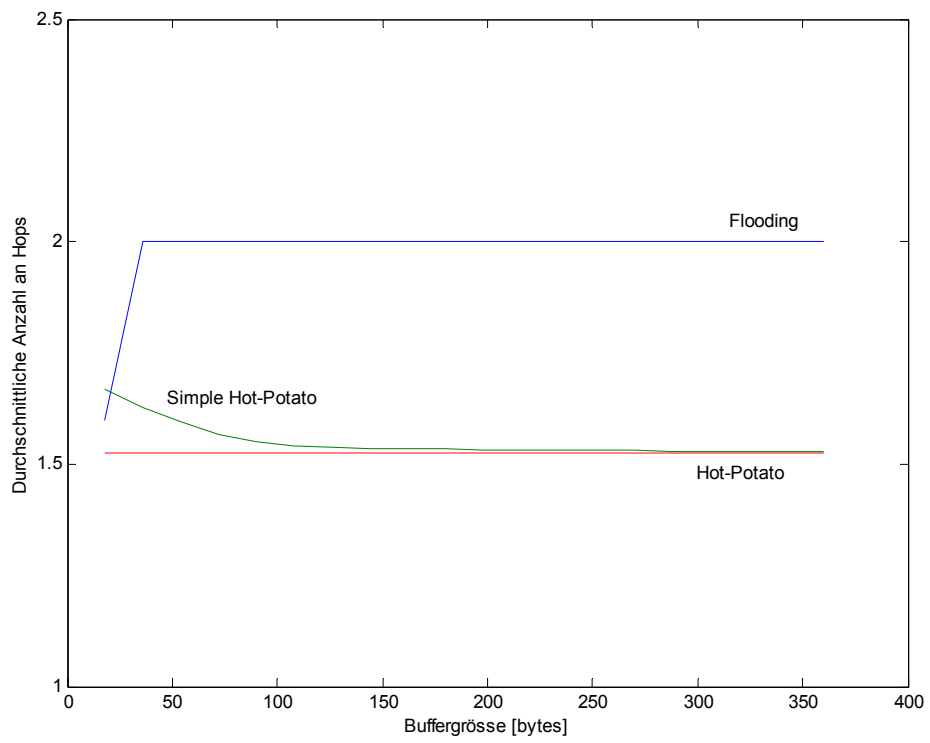
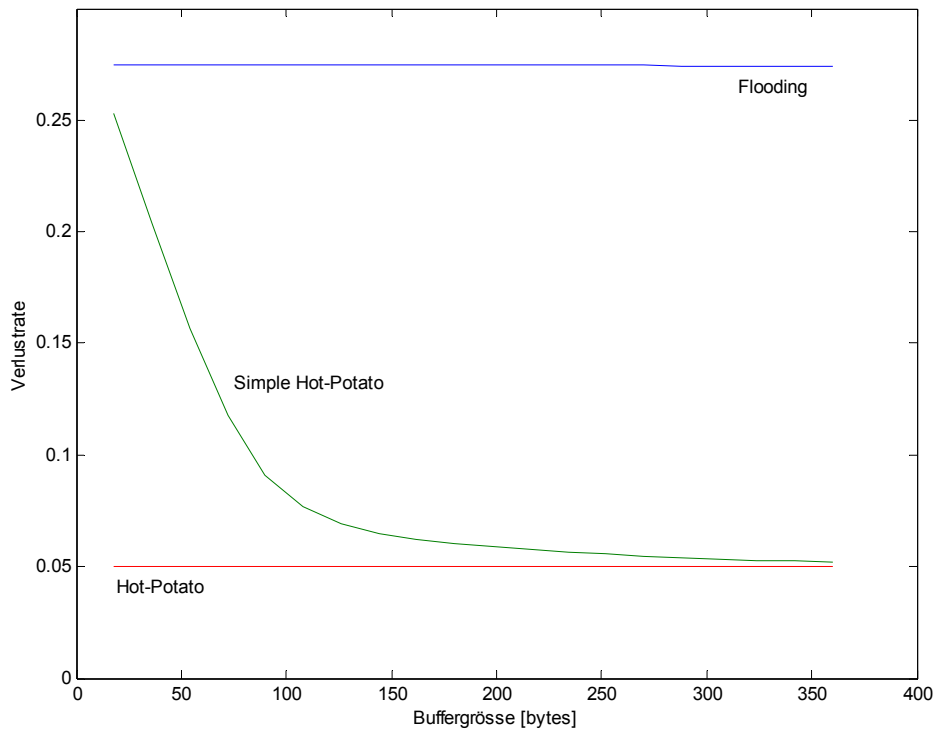


Abbildung 12 Metriken für Netzwerkfunktion 'NetworkFunc1'

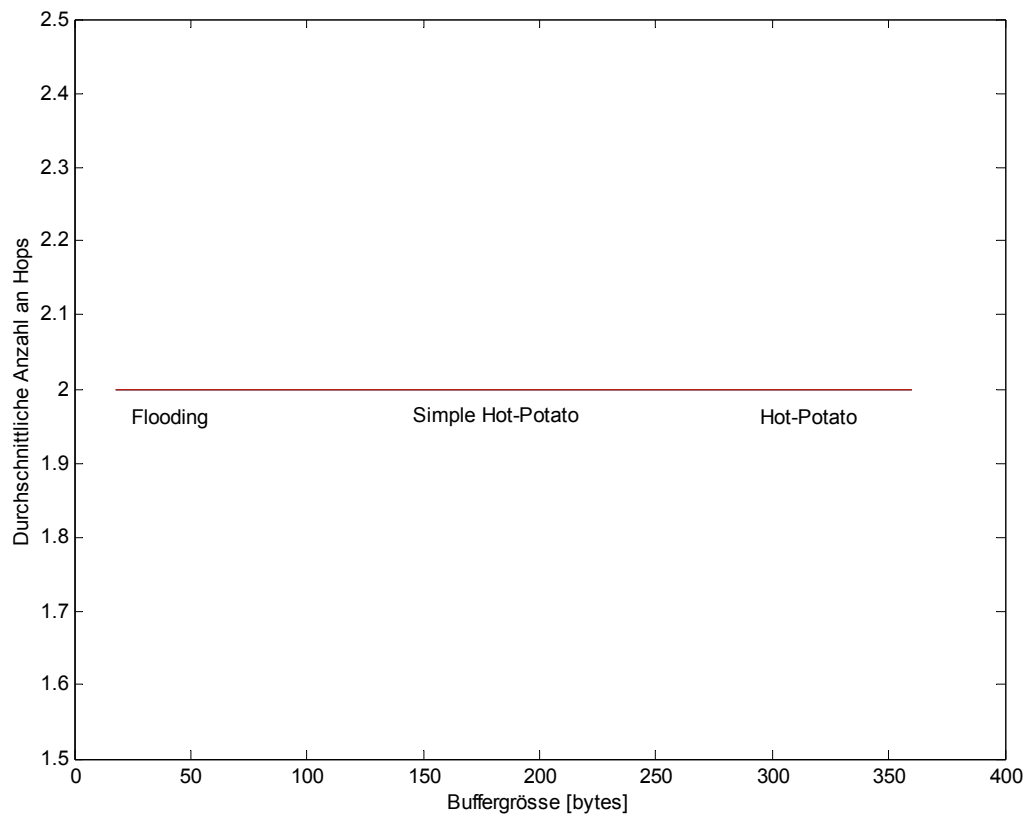
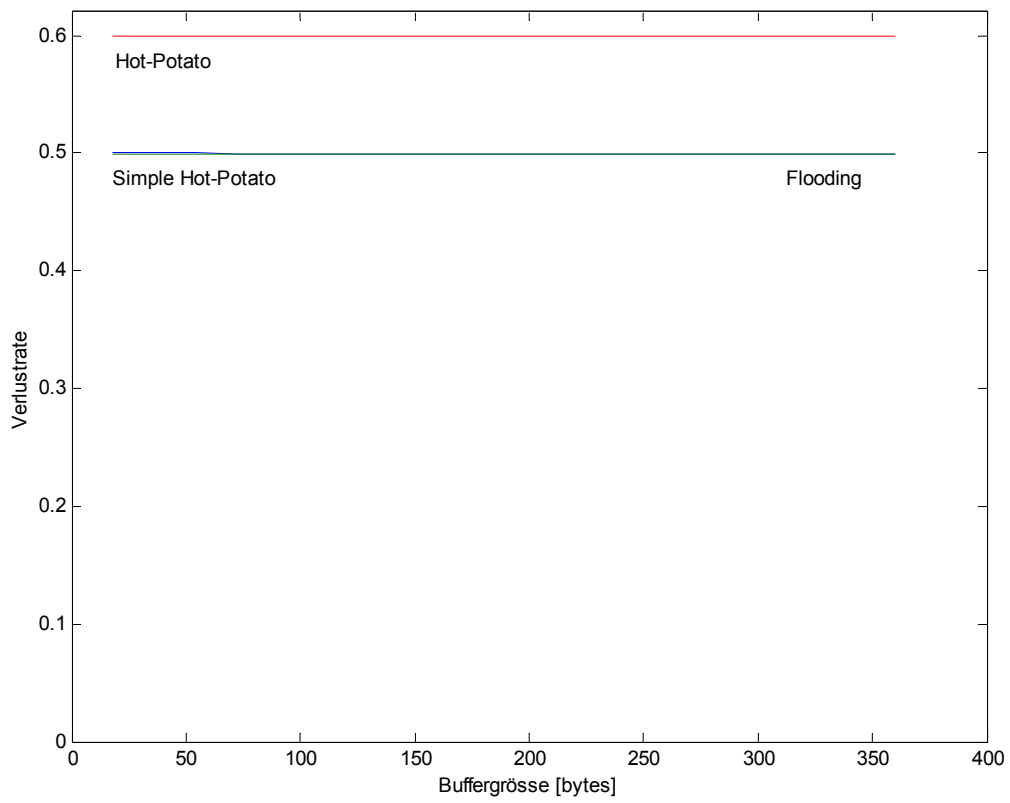


Abbildung 13 Metriken für Netzwerkfunktion 'NetworkFunc2'

Teilvermaschte Topologie

In der teilvermaschten Topologie unterliegt der Link zwischen Station S1 und Station S2 der Dynamik, die durch die Netzwerkfunktionen in Tabelle 3 beschrieben werden. Für die erste Netzwerkfunktion, die sehr langsam und gleichförmig den Linkstatus ändert, veranschaulicht Abbildung 12 die Metriken Verlustrate und Weglänge. Dieser Fall wird sehr ausführlich diskutiert, da er das typische Verhalten darstellt.

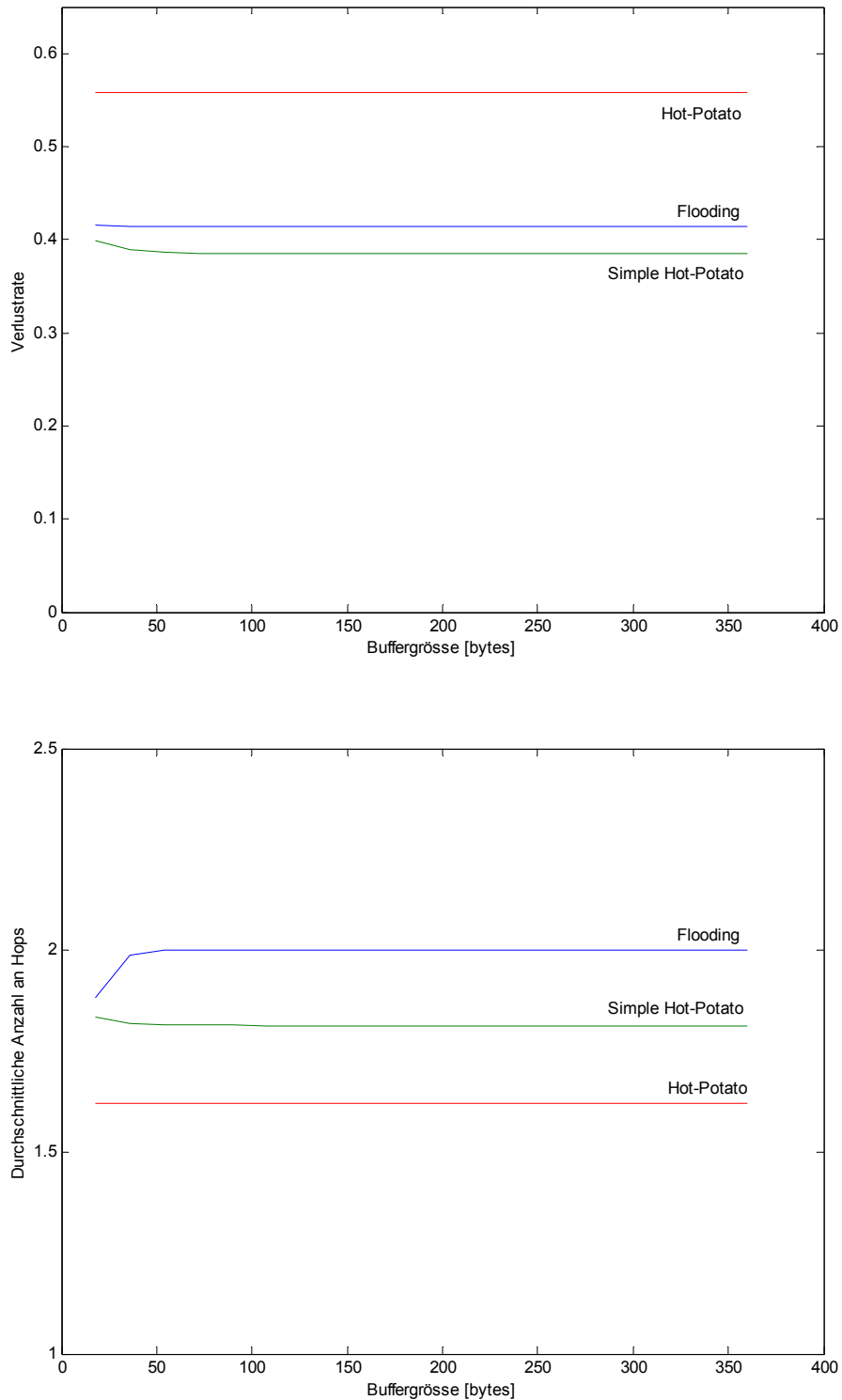


Abbildung 14 Metriken für Netzwerkfunktion 'NetworkFunc3'

Es ist sofort zu sehen, dass Hot-Potato mit dieser Dynamik sehr gut umgehen kann. Die Verlustrate ist bei einer sehr kleinen Buffergrösse äusserst gering. Steht der S1-S2 Link nicht zur Verfügung, dann läuft der entsprechende Sendebuffer voll. Folglich, entscheidet der Algorithmus, den anderen Link zu benutzen, der keinen Verlust aufweist. Der Anzahl der Hops bestätigt, dass in der Hälfte alle Entscheidungen der Weg über Station S3 gewählt wurde. Die trotzdem vorhandene Verlustrate wird von den Paketen gebildet, die verloren gehen, während sie gerade vom S1-S2 Link transportiert werden, wenn dieser gerade versagt. Flooding dagegen schneidet schlecht ab. Es unterscheidet keinen Buffer, sondern sendet an alle. Damit auch an den bereits gefüllten, womit Pakete verworfen werden. Die hohe Verlustrate erklärt sich durch Einbeziehung der Mehrarbeit (Overhead), der durch die vielen Duplikate entsteht. Aber alle Pakete werden auch über Station S3 geroutet. Wenn dann der Link S1-S2 ausfällt, waren die Pakete, die an S2 adressiert sind, schon 2 Hops unterwegs. Ist der Link wieder verfügbar, dann sind die Pakete aus dem Buffer nur noch Duplikate für S2 und werden verworfen. Angenommene Pakete, wurden deshalb fast ausschliesslich über S3 geroutet, und waren damit 2 Hops unterwegs, was die Abbildung zeigt. Simple Hot-Potato hat zu Beginn fast die gleiche Verlustrate wie Flooding. Die Verlustrate kann aber gesenkt werden bis fast auf das Niveau von Hot-Potato. Das hängt mit den vergrösserten Sendebuffern zusammen. Sie können den Effekt von fehlgerouteten Paketen dämpfen und sie erst schicken, wenn der Link wieder vorhanden ist. Das ist auch bei Flooding der Fall. Aber im Gegensatz zu Flooding produziert Simple Hot-Potato keine Duplikate, die wertvollen Platz in den Buffern belegen. Da die Pakete gleichmässig auf die Links verteilt werden und die Verlustrate klein ist, wird fast die Hälfte aller Pakete über den S3-S2 Link transportiert, während die andere Hälfte über den S1-S2 Link nach S2 gelangt. Das erklärt die durchschnittlich Anzahl an Hops von ca. 1.5. Die Abbildung 13 und die Abbildung 14 zeigen die Metriken diesmal unter der Verwendung von NetworkFunc2 und NetworkFunc3 aus Tabelle 3.

Zu erwarten wäre eigentlich ein ähnliches Bild wie in Abbildung 12, vielleicht mit höheren Verlustraten, aber mit ähnlichen Verläufen der Kurven. Überraschenderweise schneidet nun Hot-Potato am schlechtesten ab. Mit NetworkFunc2 ist die Änderung des Linkstatus so schnell, dass kein Paket mehr darüber transportiert werden kann. Alle an S2 angekommenen Pakete waren über S3 geroutet worden. Das zeigt deutlich die zurückgelegte Weglänge. Das schnelle Verlieren der Pakete bewirkt, dass der Sendebuffer des S1-S2 Links schnell geleert wird. Damit wird dieser Buffer aber von Hot-Potato Protokoll bevorzugt und überdurchschnittlich viele Pakete gehen verloren. Ein ähnliches Verhalten zeigt sich unter Verwendung von NetworkFunc3. Auch hier ist die Änderung des Linkstatus schnell, allerdings langsamer als bei der Funktion zuvor. Damit können Pakete über den S1-S2 Link übertragen werden, was die Anzahl der Hops senkt und auch die Verlustrate.

Partitionierung

Im diesem Fall werden die Netzwerkfunktionen aus Tabelle 4 benutzt. Die Abbildung 15, die Abbildung 16, die Abbildung 17 und die Abbildung 18 veranschaulichen die Metriken für diese Netzwerkfunktionen.

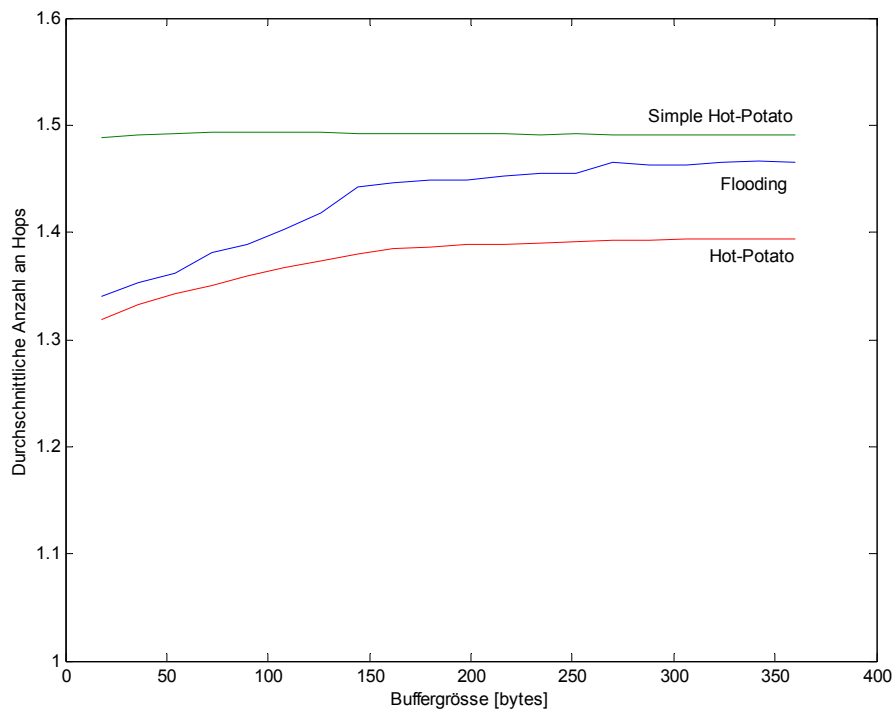
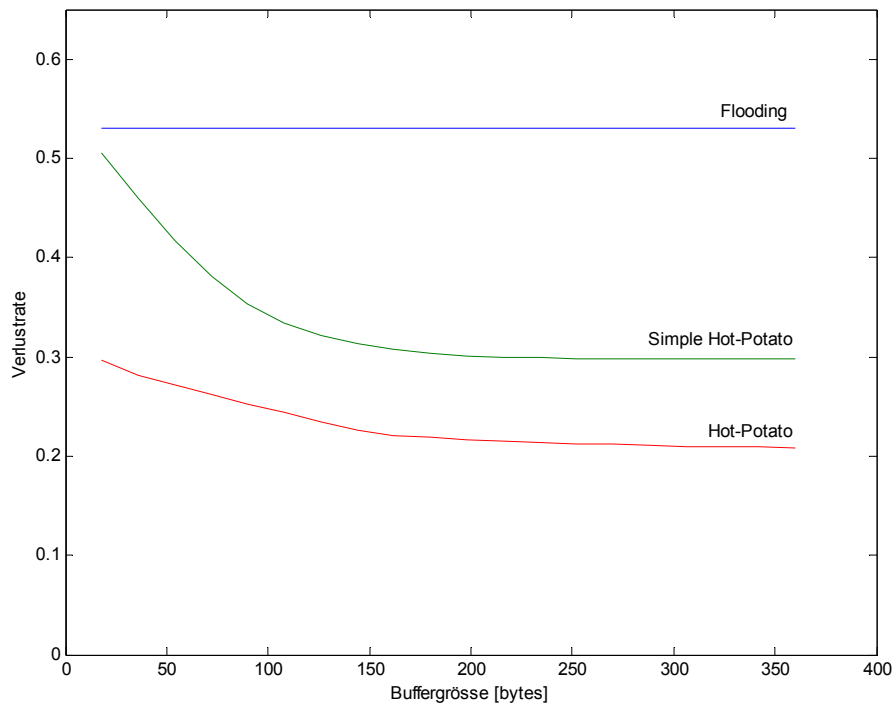


Abbildung 15 Metriken für Netzwerkfunktion 'NetworkFunc1'

Der Verlauf der Verlustrate ist ähnlich dem Verlauf in Abbildung 12. Da die Änderungsgeschwindigkeit der Links recht niedrig ist, ist das der Charakteristik der NetworkFunc1 in Tabelle 3 ähnlich. Allerdings sind hierbei die Verlustraten wesentlich höher. Das liegt daran, dass jetzt zusätzlich der S3-S2 Link ein dynamisches Verhalten zeigt. Beide Links S1-S2 und S3-S2 haben im Durchschnitt die selbe Zuverlässigkeit, allerdings wird der Weg über den S1-S2 Link bevorzugt.

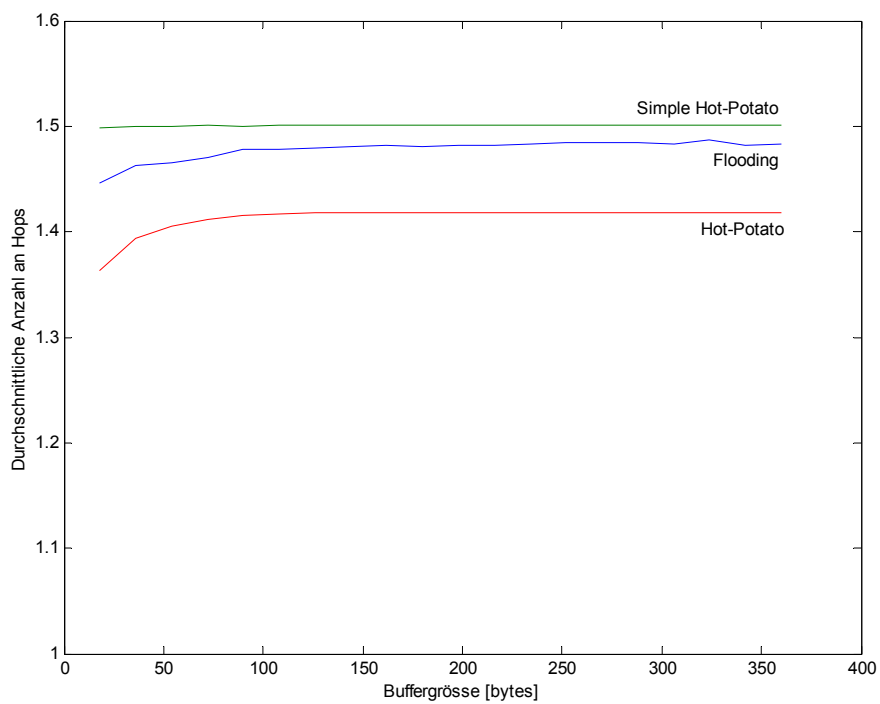
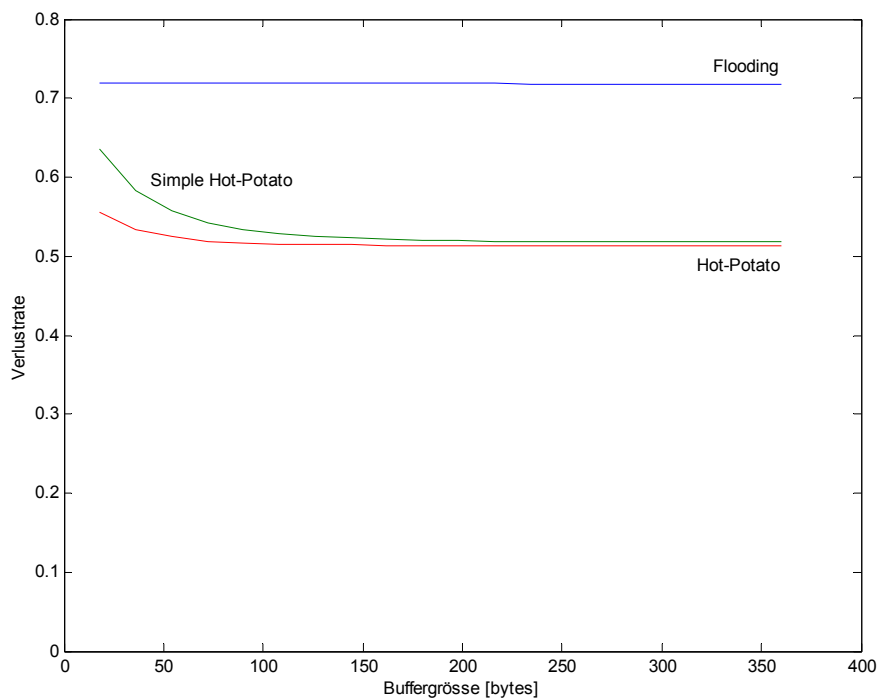


Abbildung 16 Metriken für Netzwerkfunktion 'NetworkFunc2'

In Abbildung 16 ändern sich beide Links nach der NetworkFunc2 aus Tabelle 4. Beide Links untereinander weisen die gleiche Zuverlässigkeit auf, aber sie sind dynamischer als unter NetworkFunc1. Aufgrund der höheren Dynamik ergibt sich eine höhere Verlustrate. Aufgrund der Zuverlässigkeit ist der Verlauf der Anzahl der Hops ähnlich dem in Abbildung 15.

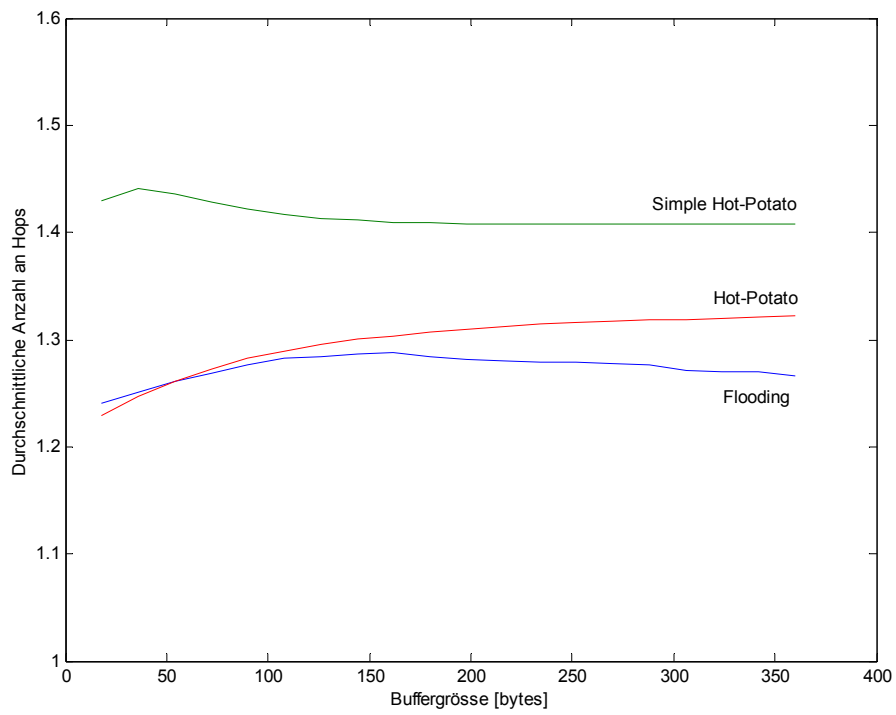
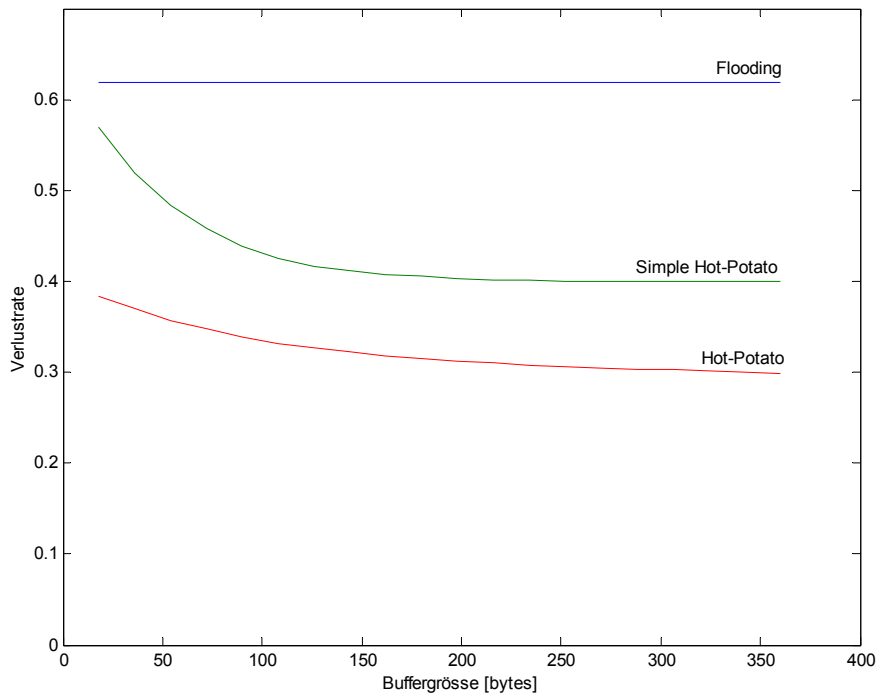


Abbildung 17 Metriken für Netzwerkfunktion 'NetworkFunc3'

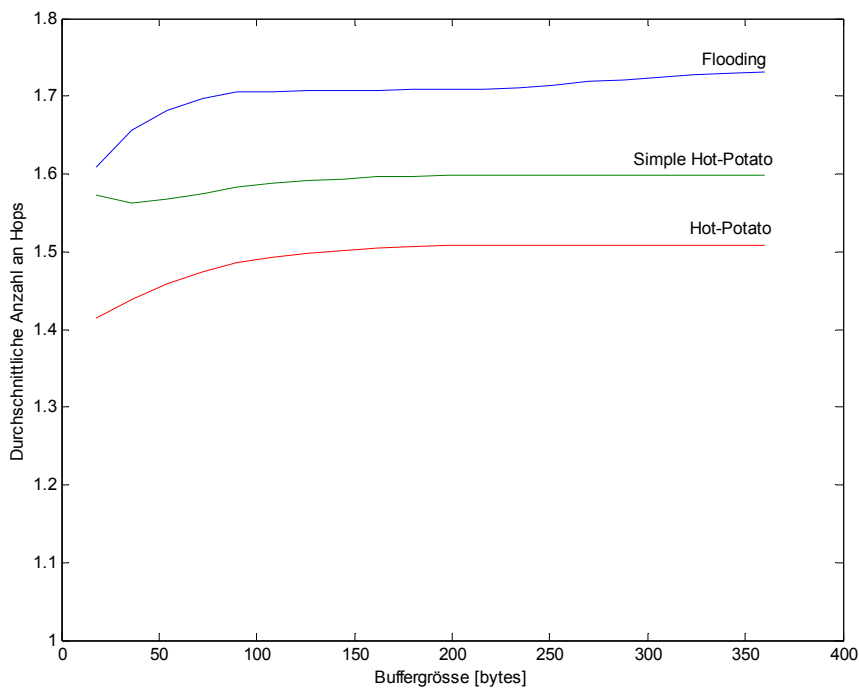
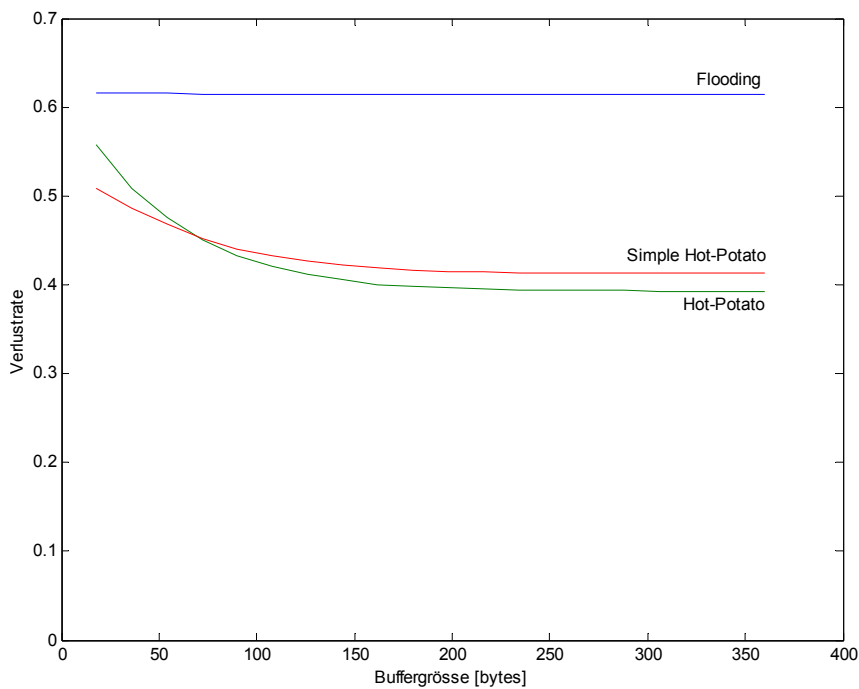


Abbildung 18 Metriken für Netzwerkfunktion 'NetworkFunc3'

Interessant an den Abbildungen 17 und 18 ist die Ähnlichkeit mit den Abbildungen zuvor. Die Verlustrate von Abbildung 17 ähnelt sehr stark dem Verlauf in Abbildung 15, während der Verlauf in Abbildung 18 dem in Abbildung 16 ähnelt. Die Gemeinsamkeit in denen sich ähnelnden Abbildungen ist das Verhalten des S1-S2 Links. In allen Fällen lag der gleiche Algorithmus für diesen Link zugrunde. Das zeigt, dass das Verhalten dieses Links, der den kürzesten Weg zwischen Quelle und Ziel markiert, für den Verlauf der Verlustrate bestimmend ist. Allen Verlustraten ist dabei gemein, dass eine Erhöhung der Buffergröße über 200 byte keine nennenswerte Verringerung bringt, sondern sich eine Sättigung einstellt.

4.2.3 Simulation - Zusammenfassung

Dynamik verändert drastisch die Leistungsfähigkeit des Netzwerks. Während im statischen Fall kein Paketverlust zu verzeichnen war, stieg er bei sich verändernden Bedingungen drastisch an (teilweise über 70%). Routing Protokolle, wie Hot-Potato, die Routing Entscheidungen von Bufferfüllständen abhängig machen, können langsame gleichförmige Änderungen gut kompensieren. Auch die Grössen der Sendebuffer tragen dazu bei. Bei sehr schnellen Änderungen aber, bricht diese Leistung des Hot-Potato Protokolls ein, da die Sendebuffer aufgrund immer wieder abbrechender Links sehr schnell geleert werden. Das Hot-Potato Protokoll misinterpretiert das als Links mit einer hohen Bandbreite und plaziert weitere Datenpakete in die Sendebuffer.

Paketverlust kann auch durch grössere Sendebuffer reduziert werden. Dabei stellte sich heraus, dass eine Erhöhung über einen bestimmten Wert keine nennenswerte Verbesserung der Verlustrate herbeiführte, sondern eine Sättigung eintrat.

Alle Ergebnisse wurden unter Verwendung von Wahrscheinlichkeitsfunktionen erzielt, die das Verhalten des Netzwerkes modellieren.

5 Zusammenfassung und Ausblick

Der vermehrte Einsatz und Gebrauch von persönlichen Geräten, die so klein sind, dass man sie problemlos in Kleidung einbetten kann, führte zu Überlegungen, wie man diese Geräte miteinander kommunizieren lassen kann. Eine Vielzahl von neuen Applikationen und Erweiterungen bestehender sind denkbar, wie einige Beispiele in der Einleitung zeigten. Dabei eingesetzte Netzwerke unterliegen restriktiven Bedingungen, wie begrenzten Hardwareressourcen oder langsamen und unzuverlässigen Verbindungen. Daher wurden Routing Verfahren eingesetzt, mit dem Ziel, die Leistungsfähigkeit solcher Netzwerke zu erhöhen.

In dieser Arbeit wurde nach einer Analyse verschiedener Routing Verfahren die Routing Protokolle Flooding, Hot-Potato und Simple Hot-Potato ausgewählt und hinsichtlich ihres Ressourcenbedarfs nach Bandbreite und Bufferkapazität und ihres Verhaltens in dynamischen Netzwerken untersucht. Gegenüber dem Ansatz des Flooding Protokolls, die Pakete einfach an alle Abgänge zu verteilen, zeigten andere Algorithmen wie Hot-Potato wesentlich bessere Leistungen. Das Hot-Potato Protokoll zeigte sehr gute Leistungen hinsichtlich des Ressourcenbedarfs, aber versagte bei hoher Dynamik des Netzwerkes. Die beiden anderen Protokolle schnitten in diesen Fällen besser ab. Es wird daher vorgeschlagen, Hot-Potato um eine Flooding-Funktion zu erweitern, die bei hoher Dynamik des Netzwerkes eingesetzt wird. Schwierig wird dabei die Erkennung dieser Dynamik, da keines der Protokolle direkten Zugriff auf den Linkstatus hat. Diese Funktion könnte allerdings hinzugefügt werden. Eine statistische Erkennung über Bufferfüllstände ist ebenfalls denkbar. Der Nachteil dabei ist erhöhter Speicherverbrauch und Rechenaufwand. Als interessante Alternative bietet sich Simple Hot-Potato an, welches die Pakete nach dem Zufallsprinzip verteilt. Der Ressourcenbedarf ist zwar grösser als der des Hot-Potato, aber kleiner als der des Flooding Verfahrens. In Situationen schneller Veränderungen im Netzwerk weist dieses Protokoll die geringste Verlustrate auf. Neben der Modifizierung des Hot-Potato Verfahrens, wird als zweite Möglichkeit ein hybrides Protokoll vorgeschlagen, welches zum einen wie Hot-Potato funktioniert, aber unter bestimmten Situation sich wie Simple Hot-Potato verhält. Problem ist wiederum die Erkennung des Zustandes, der die Verhaltensänderung auslöst. Zudem muss die Station dann zwei Routing Protokolle implementieren, was den Speicherbedarf der Treiberhardware erhöht.

Es wurde gezeigt, dass der Einsatz von Routing Protokollen zu einer Verbesserung der Leistungsfähigkeit von Netzwerken wie FAN führt. Folgende Arbeiten könnten sich darauf konzentrieren, auch den Bedarf der Ressourcen Energie, Speicherplatz und Rechenleistung in die Simulation mit einzubeziehen, da diese die Voraussetzung dafür sind, auch auf Kleinstgeräten komplexere Routing Algorithmen ablaufen zu lassen. Desweiteren sollte auch versucht werden, Eigenschaften wie die Bandbreite der FAN Links zu verbessern. Protokolle mit einem höheren Bedarf an Bandbreite und komplexeren Eigenschaften werden dann interessant. Ausserdem sollten alternative Möglichkeiten, um zwei Kleidungsstücke miteinander zu verbinden, erforscht werden. Die FAN-Simulations-Architektur ist so flexibel gestaltet, auch andere Formen der Verbindung als über elektromagnetische Felder zu simulieren. Umfangreiche Tests von Prototypen sollen in Zukunft Parameter liefern, mit denen die Simulation konfiguriert werden kann. Dadurch kann die Zweckmässigkeit in realen Situationen besser evaluiert werden.

6 Literaturverzeichnis

- [Baran 1964] Baran, P., 1964. On Distributed Communications Networks. *IEEE Transactions on Communications*. pp.1-9
- [Calvert, Doar & Zegura, 1997] Calvert, K., Doar, M., and Zegura, E. W. June 1997. Modeling Internet Topology. *IEEE Communications Magazine*. 35(Jun), pp.160-163.
- [Davis et al., 2001] Davis J. II, Hylands C., Kienhuis B., Lee E.A., Liu J., Liu X., Muliadi L., Neuendorffer S., Tsay J., Vogel B., Xiong Y. 2001. *Heterogeneous Concurrent Modeling and Design in Java*. [Online]. University of California, Berkeley, CA USA. Available from: <http://ptolemy.eecs.berkeley.edu/ptolemyII/> [Accessed 09/14/01].
- [Henderson & Sahouria] Henderson T., Sahouria E. *Web traffic generator* [Online]. Available from: <http://www.cs.berkeley.edu/~tomh/> [Accessed 09/29/01]
- [Hum, 2001] Hum, A.P.J. 2001. Fabric area network - a new wireless communications infrastructure to enable ubiquitous networking and sensing on intelligent clothing. *Computer Networks*. 35, pp.391-399
- [Johnson & Maltz, 1996] Johnson, D.B., and Maltz, D.A. 1996. *Mobile Computing*. chapter 5, pp. 153-181, Kluwer
- [Malkin, 1998] Malkin, G., editor. RIP Version 2. Internet Request for Comments RFC2453, November 1998
- [Moy, 1998] Moy, J., editor. OSPF Version 2. Internet Request for Comments RFC2328, April 1998.
- [ns-2, 2001a] University of California at Berkeley, CA, USA. *The Network Simulator – ns-2* [Online]. Available from: <http://www.isi.edu/nsnam/ns/> [Accessed 09/29/01].
- [ns-2, 2001b] *Research using ns* [Online] Available from: <http://www.isi.edu/nsnam/ns/ns-research.html> [Accessed 09/29/01]
- [Padridge et al., 2000] Kurt Padridge, Larry Arnstein, Gaetano Boriello, Turner Whitted. Fast Intrabody Signaling, 3rd *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA) Monterey, 7-8 December, 2000*
- [Sonnerstam, 1999] D. Sonnerstam (Hrsg.) *Specification of the Bluetooth System Version 1.0 B*, Bluetooth SIG; Dezember 1999
- [Weiser, 1991] M. Weiser. *The Computer for the 21st Century*, Scientific American, September 1991 Vol. 265 No. 3, pp 94-104