Diplomarbeit Nr. 2256

Entwurf und Evaluation einer knotendisjunkten Pfadsuche in s-verbundenen Graphen

Mirko Knoll

Studiengang :	Informatik
Prüfer :	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Betreuer :	Dipl.-Inf. Arno Wacker
Beginn am :	1. September 2004
Beendet am :	8. März 2005
CR-Nummer :	C.2.4, C.4, C5.3, D.4.5, D.4.8

IPVS

Institut für Parallele und
Verteilte Systeme (IPVS)
Abteilung Verteilte Systeme
Universitätsstraße 38
70569 Stuttgart

Kurzgefaßt

Das 3PC (Peer to Peer Pervasive Computing) Projekt hat es sich zur Aufgabe gemacht, einen Rahmen zur Interaktion einer Vielzahl heterogener Geräte zu schaffen. Das Sortiment umfasst dabei die verschiedensten Geräte, vom Kleinstsensor aus Alltagsgeräten bis hin zu leistungsstarken Servern. Die Kommunikation der Geräte erfolgt durch Nutzung aktueller Funkstandards wie Bluetooth, IEEE 802.11 oder weiteren kabellosen Verfahren wie IrDA. In ihrer Funktionalität ergänzen sich die Geräte untereinander, um dem Benutzer den größtmöglichen Nutzen zu liefern. Im 3PC Projekt wird das Gebiet der Heimautomatisierung untersucht. Ein Temperatur-Sensor übermittelt seine Daten beispielsweise an die Ventilsteuerung der Radiatoren um die Raumtemperatur anzupassen.

Drahtlose Kommunikation bedarf Sicherheitsmechanismen, um sich gegen Angreifer von außen zu schützen. Die Voraussetzungen um so eine Umgebung zu schaffen, werden durch das G-SPAN Projekt untersucht. Durch die minimalen zur Verfügung stehenden Ressourcen in Kleinstgeräten eignen sich komplexe kryptographische Verfahren nicht. In G-SPAN wird daher versucht, durch eine spezielle Netzwerktopologie Ressourcen-schonende Protokolle zu entwickeln um das Netzwerk bestmöglich zu schützen. In dieser Arbeit wurden ein neues Protokoll und Testszenarien entwickelt, die es ermöglichen bestehende und zukünftige Verfahren zu testen und zu vergleichen.

Inhaltsverzeichnis

1 **Einführung** **1**

1.1 Motivation 1

1.2 Hintergrund 2

1.3 Struktur der Arbeit 2

2 **Grundlagen** **3**

2.1 Sensornetzwerke 3

2.2 G-SPAN 5

3 **Problemstellung** **9**

4 **Szenarioanalyse** **11**

4.1 Sensortypen 11

4.2 Aktoren 15

4.3 Übersicht 16

4.4 Fazit 17

5 **Algorithmen** **19**

5.1 Maxflow 19

5.1.1 Die Ford/Fulkerson Methode 23

5.1.2 Laufzeit 25

5.2 Die Tiefensuche 27

5.3 Rekursiver Schlüsselaufbau (RKEP) 29

6 **Implementierung** **33**

6.1 Kritische Erfolgsfaktoren 34

6.2 Basis–Funktionen 37

6.3 Maxflow (LSP) 37

6.3.1 Setup 40

6.3.2 Pfadsuche 44

6.3.3 Nachrichtenaufwand 49

6.3.4 Speicherverbrauch 50

6.4 Maxflow (GV) 51

6.4.1 Setup 53

6.4.2 Pfadsuche 57

6.4.3 Nachrichtenaufwand 57

6.4.4 Speicherverbrauch 58

6.5 Tiefensuche 59

6.6 RKEP (manuell) 61

6.6.1 Nachrichtenaufwand 62

6.6.2 Speicherverbrauch 63

6.7 RKEP (automatisch) 64

6.7.1 Nachrichtenaufwand 64

6.7.2 Speicheraufwand 65

7 **Analyse** **67**

7.1 Testszenarien 67

7.2 Evaluation 69

8 **Resümee** **77**

8.1 Zusammenfassung 77

8.2 Ausblick 78

A **Die Administrations-Software** **81**

Grundfunktionen 81

Erweiterung 83

B **Referenzen** **87**

Abbildungsverzeichnis

Abbildung 1 - Beispielumgebung für den Einsatz von Sensoren [20] 11
Abbildung 2 - Maximaler Fluss in einem Netzwerk 20
Abbildung 3 - Bestimmung des maximalen Flusses 24
Abbildung 4 - Die Tiefensuche 27
Abbildung 5 - Paketverhalten in RKEP (s=2) 29
Abbildung 6 - fehlerhafter Graphaufbau für RKEP 30
Abbildung 7 - RKEP s – Connector 31
Abbildung 8 - maximale Entfernung (links: kettenförmig, rechts: zufällig) 34
Abbildung 9 - Aufbau eines Link State Paketes für das Maxflow Protokoll 38
Abbildung 10 - Aufbau des Netzwerks unter Maxflow (LSP) 40
Abbildung 11 - Auswirkung eines Master Resets 42
Abbildung 12 - Pfadsuche in Maxflow (LSP) 44
Abbildung 13 - Graph Transformation (links: ungerichtet, rechts: gerichtet) 45
Abbildung 14 - Modifikation zur Einsparung von Kanten 46
Abbildung 15 - Aufbau von Schlüsselpaketen 47
Abbildung 16 - Global View Paketaufbau 51
Abbildung 17 – GV Done Paketaufbau 52
Abbildung 18 – Flussdiagramm Maxflow (GV) 53
Abbildung 19 - RKEP Kette 63
Abbildung 20 - SecAdmin V2.8 81
Abbildung 21 - SecAdmin V3.7.5 83

Abkürzungsverzeichnis

3PC	Peer to Peer Pervasive Computing
BS	Breitensuche
DOS	Denial of Service
G-SPAN	Graph Based Security Protocols for Ad-Hoc Networks
GV	Sicht auf das Netzwerk (Global View)
LSP	Link State Packet
m	Anzahl Kanten
n	Anzahl Knoten
p	Anzahl Pfadsuchen
r	Maximal defekte Knoten
RKEP	Recursive Key Establishment Protocol
s	Sicherheits-Level S ohne Redundanz
trans.	transformiert
z	Sicherheits-Level mit Redundanz

1 Einführung

1.1 Motivation

Sie kommen nach einem anstrengenden Arbeitstag nach Hause. Die Haustür springt automatisch auf nachdem die Netzhautüberprüfung sie als Bewohner identifiziert hat und Sie treten ein. Auf ihrem Weg zum Wohnzimmer schalten sich automatisch dort die Lichter ein, wo Sie sie benötigen. Draußen ist es mittlerweile dunkel geworden worauf die Rollläden am ganzen Haus heruntergefahren werden und die Heizung etwas hochgedreht wird. Sie machen es sich in Ihrem Fernsehsessel gemütlich, die Beleuchtung wird gedimmt und der Fernseher schaltet auf Ihren Lieblingskanal.

Szenarien wie diese könnten sich schon in sehr naher Zukunft bei jedem von uns abspielen. Die Sensortechnik ist heute schon in der Lage, auf eine Vielzahl von Ereignissen aus der Umwelt zu reagieren: im oben genannte Beispiel wären dies Bewegungsmelder, Photodioden, Temperaturmessgeräte und Drucksensoren. Und dieses Beispiel lässt sich noch beliebig erweitern. Drahtgebundene Systeme mit derartiger Funktionalität finden sich schon seit längerer Zeit auf dem Markt [1]. Um einen hohen Freiheitsgrad in der Positionierung der einzelnen Geräte und eine einfache Erweiterung des Systems zu ermöglichen, werden zukünftige Sensoren nicht kabelgebunden, sondern per Funk kommunizieren. Ein Nachteil, der dabei zum Tragen kommt ist die Tatsache, dass Funkwellen nicht an Gebäudegrenzen enden, sondern meist noch weit darüber hinaus empfangbar sind. In unserem Szenario könnte sich das folgendermaßen auswirken:

Auf der Straße vor dem Haus empfängt ein Angreifer die Funkdaten der Sensoren mit einem Notebook. Diese Informationen ermöglichen es ihm festzustellen, wann Sie etwa das Haus verlassen oder ins Schlafzimmer gehen. Durch ein Überladen der Sensoren mit falschen Daten (*Denial of Service*) kann er dann bei minimaler Gefahr ins Haus einsteigen ohne gestört zu werden.

Es ist einsichtig, dass eine Verschlüsselung der Funkdaten unumgänglich ist um einen Diebstahl von Information zu verhindern oder wenigstens zu erschweren. Derzeit erhältliche Systeme verfügen bestenfalls über einen so genannten Hauscode, der primär nur dazu dient, das System von anderen in der unmittelbaren Nachbarschaft abzugrenzen. Damit die Sensorik auch in billigen Geräten Einzug halten kann, darf sie keine komplexe und somit teure Hardware enthalten. Für diesen Fall müssen Netzwerkprotokolle entwickelt werden, die trotz primitiver Hardware eine möglichst sichere Kommunikation ermöglichen.

1.2 Hintergrund

Immer mehr technische Geräte nehmen Einfluss auf unserer tägliches Leben. Für nahezu jeden Alltagsgegenstand existieren mittlerweile elektronische Helfer. Der Forschungsbereich des Ubiquitous Computing (UC) hat es sich zur Aufgabe gemacht, diese Geräte so zu entwickeln, dass diese kostengünstig hergestellt werden können und stellt diesen Geräten eine geeignete Infrastruktur zur Verfügung. Zur flexiblen Installation und Erweiterung hat man sich daher auf eine Kommunikation über Funk verständigt.

Der UC-Teilbereich Sensornetzwerke befasst sich mit Netzen aus einer Vielzahl Sensoren, die einen Daten-orientierten Dienst bereitstellen. Dadurch können z.B. in Gebäuden Anfragen wie: „Welche Temperatur hat es im Büro XY im 2.Stock ?“ unterstützt werden.

Das 3PC Projekt [2] beschäftigt sich ebenfalls mit Sensornetzwerken. Der Schwerpunkt liegt jedoch auf dem Gebiet der adaptiven Systeme (siehe auch [3]) und betrachtet daher hauptsächlich spontane, *peer-to-peer* basierende Netzwerke mit einer geringeren Geräteanzahl. In diesem Kontext untersucht das G-SPAN Projekt Protokolle, die für Sicherheit in diesen Netzwerken garantieren können.

1.3 Struktur der Arbeit

In Kapitel 2 werden Grundkenntnisse vermittelt, die es dem Leser ermöglichen, sich ein Bild von der Struktur und Komplexität des Szenarios zu machen. Diese sind hilfreich um die in Kapitel 3 beschriebenen Schwierigkeiten und Aufgaben nachvollziehen zu können. Das darauf folgende Kapitel schildert ein mögliches Szenario für den Einsatz von Sensoren und gibt dabei Aufschluss über die verschiedenen Typen sowie deren Einsatzgebiet. Kapitel 5 beschreibt die grundlegenden Funktionen der in dieser Arbeit behandelten Algorithmen. Das anschließende Kapitel beschreibt die Veränderungen an den Verfahren, die notwendig sind um sie für die Pfadsuche in s–verbundenen Graphen zu verwenden. In Kapitel 7 werden die entwickelten Protokolle in verschiedenen Szenarien getestet um ihr Verhalten zu untersuchen und Vergleiche zwischen ihnen vorzunehmen. Die Ergebnisse dieser Tests werden in Kapitel 8 zusammengefasst und abschließend bewertet. Im Anhang findet sich noch eine Dokumentation über die Weiterentwicklung des Programms mit dem die Messungen durchgeführt wurden.

2 Grundlagen

Dieses Kapitel gibt einen Einblick in die verschiedenen Faktoren, die beim Aufbau eines Netzwerks aus Sensoren berücksichtigt werden müssen. Anschließend wird erläutert, wie die einzelnen Punkte im G-SPAN Projekt realisiert wurden und welche Probleme dabei auftreten können.

2.1 Sensornetzwerke

Wie in Kapitel 1 gezeigt, ist es in einigen Einsatzgebieten durchaus notwendig, Daten zu verschlüsseln um es Unbefugten möglichst schwierig zu machen, Zugang zu sensiblen Daten zu erhalten. Mit der Einführung von Kryptographie stellt sich die Frage, welches der beiden Verfahren (symmetrisch oder asymmetrisch) angewendet wird. An diese Entscheidung geknüpft ist die Frage nach der Schlüsselvergabe. Diese kann dezentral (von allen Geräten gemeinsam) oder zentral (von einem dedizierten Gerät) erfolgen. Daraus ergeben sich vier verschiedene Konfigurationen:

- zentral & asymmetrisch
- zentral & symmetrisch
- dezentral & asymmetrisch
- dezentral & symmetrisch

Zentral verwaltete Netzwerke basieren auf einem Gerät (Koordinator), welches die Verwaltung der übrigen Geräte (Mitglieder) übernimmt. Dazu gehören Aufgaben wie das An- und Abmelden von Geräten sowie die Bereitstellung von Schlüsseln für die restlichen Geräte. Die übrigen Geräte können auf derartige Funktionalitäten verzichten und wenden sich hierzu an den Koordinator des Netzes. Dadurch sinkt die Komplexität der Mitglieder, was zu einfacheren und damit billigeren Geräten führt.

Verfahren, die auf einen zentralen Knoten zur Verwaltung des Netzwerks setzen leiden unter der Tatsache, dass sie allein für Verfügbarkeit des Netzes verantwortlich sind. Fällt dieser Knoten aus oder wird entwendet, so bricht das gesamte Netzwerk zusammen. Ein weiterer Nachteil ist, dass dieser Knoten allein für die Verschlüsselung der Kommunikation zwischen allen Knoten zuständig ist. Das heißt, dass für den Fall, dass ein Angreifer Zugang zu diesem Knoten erhält, die Möglichkeiten gegeben sind, den Datenverkehr mitzuhören, ohne dass dies durch andere Knoten bemerkt werden kann.

In dezentral verwalteten Netzen hingegen werden alle Aufgaben auf alle Geräte (Knoten) verteilt. Dadurch müssen diese über komplexere Funktionen wie die Schlüsselgenerierung bzw. –verwaltung verfügen, jedoch wird durch die Übernahme eines Geräts nicht das gesamte Netzwerk kompromittiert, sondern nur der Datenverkehr zu den Knoten, zu denen dieses Gerät einen Schlüssel unterhält.

Asymmetrische Kryptographieverfahren (z.B. RSA [4]) werden heute allgemein als relativ sicher empfunden. Das Verfahren beruht auf zwei unterschiedlichen Funktionen: der Verschlüsselungsfunktion V_x und der Entschlüsselungsfunktion E_x. Erstere wird in einer Datenbank öffentlich zugänglich gemacht, wohingegen letztere geheim gehalten werden muss. Für den Versand einer verschlüsselten Nachricht w sucht der Absender in der Datenbank nach der Verschlüsselungsfunktion des Adressaten. Dieser kann die Nachricht dann mit seiner Entschlüsselungsfunktion wieder dechiffrieren: $w = E_x(V_x(w))$. Der Austausch von Schlüsseln kann von den Geräten selbst (dezentral) übernommen werden. Dieses Verfahren lässt aber das Abhören durch Dritte (*man-in-the-middle*) zu. Verhindert werden kann dies nur durch eine öffentliche Zertifizierungsstelle (zentral) vermeiden. Die Geräte schicken ihren öffentlichen Schlüssel an die Zertifizierungsstelle, wo sie von dieser unterschrieben werden. Fordert ein Gerät einen öffentlichen Schlüssel eines anderen Gerätes von der Zertifizierungsstelle an, kann es dessen Authentizität prüfen. Die Verteilung der Schlüssel ist in diesem Verfahren somit recht einfach durchzuführen, jedoch wird von den Knoten eine hohe Rechenleistung gefordert, da sie Produkte aus Faktoren mit ungefähr 100 Dezimalstellen berechnen müssen. Entweder wird eine zum Teil recht lange Wartezeit für den Aufbau eines Schlüssels in Kauf genommen (allein das Entschlüsseln einer 1024-bit verschlüsselten Nachricht dauert – auf einem im Vergleich zu Sensoren leistungsstarken Palm – immer noch über 36 Sekunden [5]) oder es werden komplexere Prozessoren in den Geräten verbaut, die einerseits einen sehr hohen Energiebedarf haben und andererseits dadurch auch teurer werden.

Symmetrisch Kryptographieverfahren wie beispielsweise DES oder AES ([6]) benötigen prinzipiell eine zentrale Verwaltungsstelle um Schlüssel auszutauschen. Diese Verwaltungsstelle vergibt an beide Kommunikationspartner denselben Schlüssel. Es ist auch möglich auf die zentrale Verwaltungsstelle zu verzichten und die Schlüssel über einen sicheren Kanal - Übergabe des Schlüssels in physikalischer Form wie Disketten o. ä. – direkt (dezentral) auszutauschen. n sicheren Kanal (Übergabe des Schlüssels in physikalischer Form wie Disketten o. ä.) ausgetauscht werden. Die Komplexität der Schlüssel (sie basieren auf Verschieben und Ersetzen) ist nicht so hoch wie bei den asymmetrischen Verfahren und daher in Ressourcen-schwachen Geräten effizienter einsetzbar. Die Notwendigkeit eines sicheren Kanals zum Schlüsselaustausch erschwert jedoch den Einsatz von symmetrischen Kryptographieverfahren in ad-hoc Netzwerken.

2.2 G-SPAN

Das G-SPAN Projekt beschäftigt sich mit Sicherheitsprotokollen in ad-hoc Netzwerken. Dabei sollen Ressourcen-beschränkte Geräte unter dem Aspekt der Heimautomatisierung untersucht werden.

Die im vorherigen Abschnitt genannten Vor- und Nachteile verdeutlichen, dass eine dezentrale Netzstruktur mit einer symmetrischen Verschlüsselung für dieses Szenario die am meisten geeignete Konfiguration darstellt. Im Gegensatz zu anderen Ansätzen mit gleicher Konfiguration wie zum Beispiel *Pebblenets* (siehe [7]) geht man in G-SPAN nicht davon aus, dass die Geräte einem physikalischen Angriff standhalten können. Dies bedeutet, dass ein Angreifer, der ein Gerät in seinen Besitz bekommt, die gespeicherten Daten aus der Hardware auslesen kann. Ein derartiger Schutz vergrößert und verteuert die Geräte, weshalb in G-SPAN Lösungen angestrebt werden, die nicht auf einem derartigen Schutz aufsetzen.

In G-SPAN spielt die Skalierbarkeit der untersuchten Protokolle eine wichtige Rolle, da Netzwerke mit weit über 100 Knoten analysiert werden sollen. Für die Kommunikation untereinander verwendet jedes Paar aus Geräten einen eigenen Schlüssel. Die dabei verwendeten Sensoren verfügen jedoch nicht über ausreichend Speicher um eine große Anzahl Schlüssel zu unterhalten. Ein weiterer Nachteil dieser Vorgehensweise tritt bei der Erweiterung des Netzwerks auf. Jeder neue Knoten muss mit allen anderen Geräten einen Schlüssel teilen. Da Schlüssel über sichere Kanäle übertragen werden müssen (vorzugsweise durch eine physikalische Verbindung über Infrarot oder Kabel), bedeutet dies für den Anwender, dass er das neue Gerät physikalisch mit jedem bereits installierten Sensor verbinden muss um einen initialen Schlüssel zwischen ihnen aufzubauen. Dieser Vorgang (im Folgenden „Imprinting“ genannt) ist für größere Netze nicht tragbar und wird von G-SPAN daher vereinfacht. Anstatt Schlüssel zu allen Knoten zu speichern, unterhält jeder Knoten zu Beginn nur z Schlüssel. Dieser frei wählbare Wert stellt den Konnektivitätsfaktor des Netzwerkes dar, welcher sich aus zwei Komponenten zusammensetzt: $z = s + r$. Der Sicherheitsfaktor s ist definiert als die Anzahl der knotendisjunkten Pfade, das sind die Pfade, die außer dem Start- und Zielknoten keinen weiteren Knoten gemeinsam hat. Der Wert r definiert einen Redundanzwert. Dabei kann es sich beispielsweise um die Anzahl der zu kompensierenden Angreifer oder ausgefallenen Knoten handeln.

Das Verfahren sieht nun folgendes vor: wird die Kommunikation von Geräten, die keinen gemeinsamen Schlüssel unterhalten notwendig, so führt einer der beiden Knoten eine Pfadsuche durch. Als Resultat liefert diese dann z knotendisjunkte Pfade zum Zielknoten, über die anschließend jeweils ein Teilschlüssel geschickt wird. Die Teilschlüssel werden von Knoten zu Knoten bis zum Ziel weitergeleitet, wo sie gespeichert werden. Jeder Teilschlüssel wird dabei mit der XOR Funktion auf den vorherigen Schlüssel angewendet bis alle Teilschlüssel das Ziel erreicht haben. Somit wird sichergestellt, dass kein Knoten, der einen Teilschlüssel weitergeleitet

hat, die Kommunikation zwischen Start- und Zielknoten abhören kann, da nur diese beiden Knoten über den korrekten Schlüssel verfügen. Demnach würden auch s Pfade genügen. Gerät jedoch ein Angreifer in den Besitz eines Knotens auf jedem Pfad, so hätte er die Möglichkeit, den Schlüssel selbst zu bauen. Um nun bei einer bestimmten Anzahl Angreifer immer noch eine sichere Kommunikation zu gewährleisten, muss die Anzahl knotendisjunkter Pfade entsprechend erhöht werden. Für die Graphstruktur bedeutet die z-Verbundenheit somit, dass zwischen allen Knoten immer mindestens z knotendisjunkte Pfade vorhanden sind. Der Fokus dieser Arbeit liegt auf der Pfadsuche und nicht auf der Absicherung der Kommunikation, weshalb für folgende Betrachtungen der Wert für $r = 0$ gesetzt wird.

Alternativ zum physikalischen Imprinten bietet sich auch die Möglichkeit, dies mit einem Steuergerät (*Master Device*) vorzunehmen. Dazu wird in jedem Sensor ein individueller Master Schlüssel gespeichert, so dass das Steuergerät über diesen Schlüssel weitere Schlüssel an die Sensoren verteilen kann. Als Steuergerät können zum Beispiel Notebooks in Frage kommen. Diese verfügen über linearen Speicher und können daher auch in größeren Netzen problemlos für jeden Knoten einen Schlüssel speichern. Sind alle gewünschten Knoten imprintet, kann das Steuergerät aus dem Netzwerk entfernt werden, da es für den weiteren Betrieb nicht mehr benötigt wird. Die Verwendung eines Master Devices kann den Sicherheitslevel im Netzwerk reduzieren, da bei einer Übernahme des Steuergerätes durch Unbefugte weitere Geräte in das Netzwerk mit aufgenommen werden können. Diese spionieren dann entweder den Datenverkehr aus (passiver Angriff) oder verändern Daten (aktiver Angriff) um die Stabilität des Netzes zu verringern. Verfahren zur Identifikation aktiver Angriffs existieren zwar schon (vgl. Wachhund (*Watchdog*) [8] oder [9]) sind aber in G-SPAN bisher nicht implementiert. Es wird davon ausgegangen, dass Knoten entweder vollständig von einem Angreifer übernommen wurden oder fehlerfrei funktionieren. Für den Fall, dass ein Master Device verwendet werden soll, kann das Risiko dahingehend minimiert werden, dass die Schlüssel zur Kommunikation mit den Knoten auf einer Smartcard gespeichert werden.

Als Sensor-Hardware sollen sehr kleine und kostengünstige Mikrokontroller dienen. In früheren Arbeiten [10-12] wurde auf zwei gängige Produkte zurückgegriffen. Zum einen handelt es sich dabei um ATMega16L der Firma Atmel. Hierbei handelt es sich um einen für geringen Stromverbrauch optimierten 8-bit RISC Mikrokontroller mit einer Taktrate von 8 MHz. Als Ressource stehen ihm 16 KB Flash Speicher für Programme, 512 Byte EEPROM Datenspeicher sowie 1 KB RAM zur Verfügung. Bei dem 2. Produkt handelt es sich um den PIC18F6620 aus dem Hause Microchip. Dieser basiert ebenfalls auf einem 8-bit RISC Mikrokontroller, ist aber leistungsstärker als sein Mitbewerber. Er verfügt über einen mit 20 MHz getakteten Prozessor, 64 KB Flash Speicher, 1 KB EEPROM und 4 KB RAM. Dieser Kontroller wird häufig auf Teco Smart-Its [13] verwendet, die dem Prozessor je nach

Ausführung weiteren Speicher zur Verfügung stellen. Im Fall von G-SPAN handelt es sich um den Smart-IT Particle 2/06, welcher über einen zusätzlichen EEPROM von 32 KB Datenspeicher verfügt.

Das G-SPAN Projekt verfügt zudem über einen Simulator, in dem ein Netzwerk aus Sensoren emuliert werden kann. Dadurch können größere Versuchsaufbauten vermieden werden und neue Konfigurationen ohne Programmieraufwand der Geräte an PCs simuliert werden. Das Netzwerk wird als ungerichteter Graph $G = (V, E)$ abgebildet, wobei V für die Menge der Geräte (Sensoren) steht und E die Menge der verteilten Schlüssel darstellt. Eine Kante im $\{v1, v2\} \in E$ existiert genau dann, wenn $v1$ und $v2$ einen gemeinsamen Schlüssel aufweisen. Die Geräte befinden sich innerhalb einer partitionsfreien Kollisionsdomäne, dies bedeutet, dass alle Geräte untereinander ohne weitere Geräte (z.B. Router) miteinander kommunizieren können. Die Kommunikation im Simulator erfolgt derzeit durch Pakete im UDP Format. Ein Server-Knoten vergibt eindeutige IDs an die Knoten und verschickt Debug Nachrichten. Diese werden durch Einsatz von TCP an das Visualisierungswerkzeug (siehe Anhang A) weitergeleitet. Die Paketgröße ist je nach Protokoll und Pakettyp variabel, maximal jedoch auf 106 Bytes begrenzt.

3 Problemstellung

In diesem Kapitel soll aufgezeigt werden, welche Fragen im Bezug auf die Pfadsuche noch offen sind und welche Ansätze dabei untersucht werden. Dabei wird kurz erläutert, welche Arbeiten sich mit diesem Thema befasst haben und wodurch sich diese Arbeit von ihnen abgrenzt.

Im G-SPAN Projekt wird ein Netzwerk aus Sensoren, wie es im Bereich der Heimautomatisierung realisiert werden könnte, nachgebildet. Dabei wurden bisher jedoch keinerlei Untersuchungen angestellt, welche Anzahl von Sensoren in diesem Bereich üblich beziehungsweise realistisch ist. Diese Information ist wichtig, da sie den Bereich vorgibt, in dem die verwendeten Protokolle zur Datenübertragung effizient arbeiten müssen. Normalerweise werden Algorithmen durch die *O*-Notation verglichen. In diesem Fall kann es aber vorkommen, dass ein Protokoll trotz einer asymptotisch höheren Obergrenze aufgrund einer kleineren Konstante ein geringeres Datenvolumen aufweist als ein Protokoll mit einer asymptotisch kleineren Obergrenze.

Die Suche multipler Pfade zwischen Knoten in einem Graphen wurde schon vielfach untersucht. Oftmals handelt es sich dabei um Algorithmen, die kantendisjunkte Pfade aufzeigen [14-18] um den Verlust einer Verbindung zwischen zwei Geräten zu kompensieren. Ebenso gibt es einige Ansätze, die knotendisjunkte Pfade suchen, jedoch Voraussetzungen an den Graphen stellen, wie zum Beispiel Planarität (vgl. [19]). In den Fällen $z \leq 2$ könnten diese Algorithmen Verwendung finden. Der Wert von z soll jedoch je nach benötigter Sicherheitsstufe variabel sein, weshalb in diesem Projekt nur Verfahren in Frage kommen, die keine Einschränkungen an die Graphstruktur stellen.

Selbst nach dieser Voraussetzung existieren noch mehrere Klassen von Algorithmen, die zur Lösung dieses Problems in Frage kommen. In dieser Arbeit sollen daher stellvertretend für jeweils einen spezifischen Typ, Algorithmen bezüglich ihrer Komplexität und ihrer Eignung zur Verwendung in G-SPAN untersucht werden. Hierzu müssen Kriterien gefunden werden, anhand derer die ausgewählten Verfahren in verschiedenen Szenarien miteinander verglichen werden können. Eine theoretische Analyse soll Aufschluss über die Tauglichkeit und Effizienz liefern. Durch die Nachteile der *O*-Notation müssen ihre Ergebnisse im G-SPAN Simulator validiert werden.

4 Szenarioanalyse

In diesem Kapitel wird Aufschluss darüber gegeben, welche verschiedenen Klassen von Sensoren zur Verfügung stehen und in welcher Anzahl. Dadurch ist es möglich, eine Abschätzung vorzunehmen wie viele Geräte zur Laufzeit miteinander kommunizieren. Dieser Wert ist wichtig, da die maximale Anzahl an Teilnehmern den Bereich vorgibt, in dem die zu vergleichenden Algorithmen effizient arbeiten müssen.

4.1 Sensortypen

Es existiert eine Vielzahl unterschiedlichster Sensortypen, die sich im alltäglichen Leben kaum noch wegdenken lassen. Viele Sensorensysteme, zum Beispiel das ABS oder ESP in der Fahrzeugtechnik unterstützen uns bei unseren Aufgaben und haben unser Leben dabei angenehmer und sicherer gemacht. Doch auch in anderen Lebensbereichen ist der Einsatz von Sensoren denkbar und erstrebenswert. Die folgende Abbildung sowie die anschließende Klassifikation veranschaulichen die verschiedenen Sensortypen, die für das Gebiet der Heimautomatisierung denkbar sind.

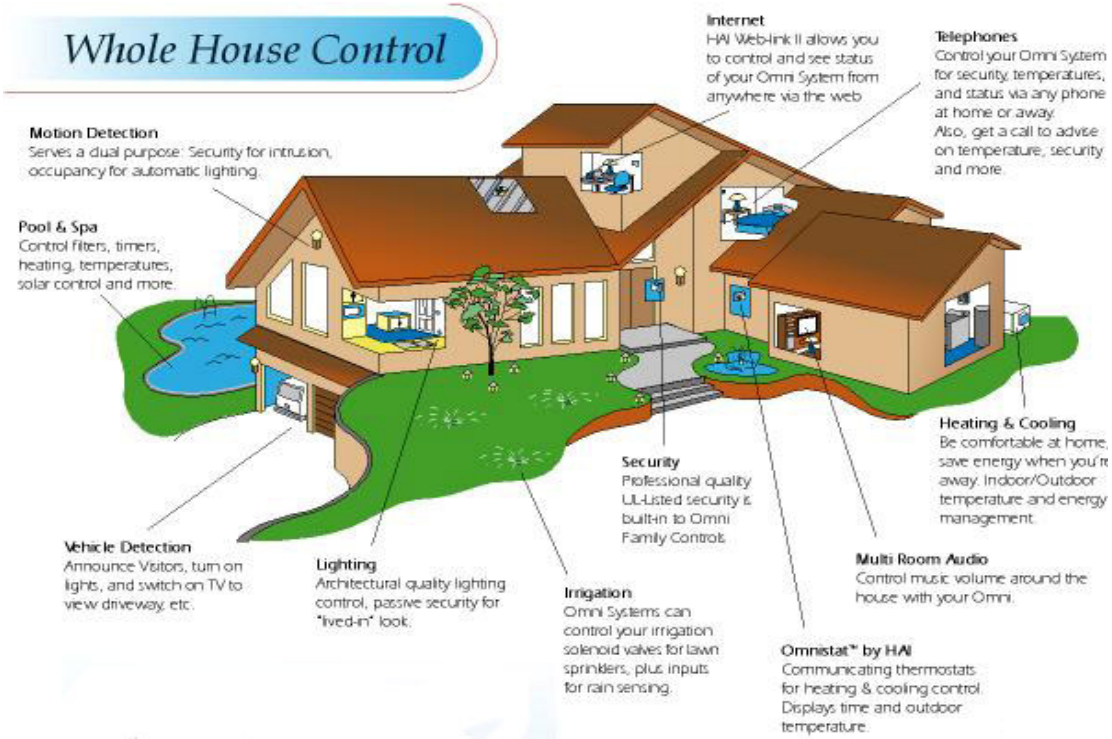


Abbildung 1 - Beispielumgebung für den Einsatz von Sensoren [20]

Bewegungssensoren

Ein Bewegungsmelder nutzt den pyroelektrischen Effekt aus, um Bewegungen zu registrieren. Vorbeilaufende Menschen oder Tiere strahlen Körperwärme aus. Diese bewirkt Temperaturunterschiede in den Kristallen des Sensors und bewirkt eine Ladungstrennung. Die daraus resultierende Spannungsdifferenz kann mit Hilfe von Elektroden an der Oberfläche des Kristalls abgegriffen werden [21]. Mit Hilfe dieser Sensoren ist es beispielsweise möglich, die Lichtsteuerung zu übernehmen. Somit wird beim Betreten des Raumes automatisch das Licht eingeschaltet und beim Verlassen des Raumes auch wieder ausgeschaltet. Des Weiteren können sie an eine Alarmanlage gekoppelt werden oder dienen als automatischer Türöffner (z.B. für Rollstuhlfahrer).

Lichtempfindliche Sensoren

In diesem Fall kommen häufig Fotodioden zum Einsatz. Diese wandeln Licht in elektrischen Strom, sobald Licht auf den p-n Übergang trifft. In der in Sperrrichtung betriebenen Diode werden bei genügender Einstrahlung von Photonen auf dem Halbleitermaterial Ladungsträger freigesetzt, die dann zu einem Stromfluss führen (vgl. [22]). Durch die Messung der Lichtstärke können viele Aufgaben automatisiert werden und es lässt sich Energie einsparen. Im Sommer können bei hoher Sonneneinstrahlung die Rollläden bzw. die Markise heruntergelassen werden um eine hohe Aufheizung der Innenräume und den Einsatz von Klimageräten zu reduzieren oder gar ganz zu verhindern. Die wohl aber naheliegendste Funktion ist das Einschalten von Licht bei Dunkelheit, z.B. durch eine Koppelung an Bewegungssensoren.

Klimasensoren

Es existiert eine Vielzahl von Effekten, die zur Bestimmung der Temperatur herangezogen werden können. Eine der gebräuchlichsten Varianten verwendet Heißleiter (*Positive Temperature Coefficient*) Widerstände. Diese bestehen aus stromleitenden Materialen, die bei hohen Temperaturen Strom besser leiten als bei tiefen und somit ihr elektrischer Widerstand sinkt (siehe [21;22]). Hygrometer messen die Luftfeuchtigkeit durch die Verwendung von hygroskopischen Materialien wie z.B. Kunsthaaren. Diese dehnen sich je nach Feuchtigkeit um 2,5 % zwischen trockener (0% Luftfeuchtigkeit) und feuchter Luft (100% Feuchtigkeit). Der wohl häufigste Einsatzzweck von Temperatursensoren liegt in der Steuerung von Heizungen oder Klimageräten. Dem Benutzer dieses Sensortyps wird so ermöglicht, immer unter angenehmsten Klimabedingungen zu leben. Des Weiteren werden diese Sensoren aber auch zur Regelung der Wassertemperatur verwendet. Temperaturmessungen entscheiden, ob das Wasser noch warm genug ist oder ob es durch den Brenner oder die Solaranlage wieder auf die gewünschte Temperatur

gebracht werden muss. Regen- und Windsensoren können genutzt werden um beispielsweise Fenster bei Regen und Stürmen zu schließen bzw. Markisen einzufahren.

Drucksensoren

Um den Druck zu messen, verwenden siliziumbasierte Geräte Dehnungsmessstreifen mit 4 implementierten Widerständen. Die Anordnung der Widerstände sorgt dafür, dass sich ihr Wert in Abhängigkeit des Drucks gegenseitig ändert (vgl. [21]). Diese Klasse von Sensoren ist derzeitig in der Heimautomatisierung noch unüblich. Trotzdem gibt es einige Ansätze, auch diese ins heimische Wohnzimmer zu bringen. Microsoft Research [23] zeigt im Projekt *Easy Living,* wie durch den Einsatz von Drucksensoren elektronische Geräte eingeschaltet werden. Vorstellbar wäre zum Beispiel die automatische Einschaltung des Fernsehers beim Hinsetzen auf den Fernsehsessel. Durch Messung des Gewichts könnten in weiteren Anwendungen auch verschiedene Personen identifiziert werden und personenspezifische Programme (z.B. automatischer Senderwechsel auf den Lieblingskanal) gestartet werden.

Feuer- / Gasmelder

Damit ein Feuermelder Alarm schlägt, muss die Konzentration von Kohlenstoffmonoxid, Kohlenstoffdioxid und anderen Verbrennungsgasen einen bestimmten Grenzwert überschreiten. Feuermelder sind heute vor allem in Räumen mit teurem Inventar schon eine Selbstverständlichkeit um dieses vor Brand zu schützen. In Wohnzimmer (z.B. in Kaminnähe) oder in der Küche sind sie gängige Lebensretter.

Rauchmelder

Diese verwenden entweder ähnliche physikalische Effekte wie Feuermelder oder arbeiten auf optischer bzw. photoelektrischer Basis mit Hilfe eines Streulichtverfahrens [21]. Dazu sendet eine Leucht- oder Laserdiode einen Lichtstrahl innerhalb des Gehäuses aus. Die Luft im Raum gelangt durch Öffnungen im Gerät in das Gehäuseinnere und kann dort registriert werden. Klare Luft reflektiert kein Licht, Rauchpartikel in der Luft dagegen schon. Der Melder schlägt Alarm sobald Streulicht auf die Fotodioden fällt, welche außerhalb des Lichtstrahls angeordnet sind. Diese Art von Sensoren ist in privaten Haushalten eher selten anzutreffen. Der primäre Einsatzzweck entspringt der Industrie um in Chemieanlagen das Austreten von giftigen Gasen festzustellen. In Wohnhäusern sind jedoch auch solche Sensoren denkbar, sofern z.B. Gasherde in der Küche zum Einsatz kommen.

Wasser- bzw. Flüssigkeitsmelder

Die Schaltung eines Wassermelders ist relativ simpel. Zwei Elektroden werden isoliert voneinander auf die zu überwachende Fläche gebracht. Läuft dort eine leitende Flüssigkeit (meist Wasser) aus, fließt Strom und der Melder schlägt Alarm. Sie eignen sich daher gut, um auslaufendes Wasser von Waschmaschinen, geplatzten Rohren oder ähnlichem zu registrieren. Komplexere Ausführungen von Flüssigkeitsmelder sind ebenfalls in anderen Teilen des Hauses denkbar. In Heizkellern kommen daher optische Melder (vgl. optische Rauchmelder) zum Einsatz um austretendes Heizöl zu registrieren.

Glasbruchmelder

Diese Art von Sensoren arbeitet auf der Basis von akustischen Signalen. Dabei wird der zu sichernde Bereich ca. 40 000 mal pro Sekunde auf glasbruchtypische Signale untersucht. Außer der Analyse der Klirrgeräusche wird zusätzlich die Druckwellenfrequenz untersucht um Fehlalarme zu vermeiden [24]. Diese Sensoren müssen dabei nicht direkt am Fenster montiert werden, so dass auch dank ihrer hohen Reichweite meist ein Sensor pro Raum ausreicht, Einbrüche zu erkennen und Gegenmaßnahmen einzuleiten.

Lesemodule (RFID)

Diese Module bestehen aus einer stromdurchflossenen Spule, die ein Magnetfeld erzeugt. Durch Annäherung eines metallischen Gegenstandes (aktive und passive Transponder) in das Magnetfeld der Spule ändert sich dort die Induktivität [21]. Die jeweiligen Änderungen der Induktivität durch Transponder mit eindeutigem Code kann genutzt werden, um dadurch verschiedene Sicherheitsanwendungen [25] zu ermöglichen. Außerhalb der Zutrittskontrolle sind durchaus auch Anwendungen in privaten Haushalten vorstellbar. Beispielsweise können durch die eindeutige Identifizierung von Personen – ähnlich wie bei Drucksensoren – benutzerspezifische Abläufe getriggert werden.

Kontaktmelder

Kontaktmelder werden am einfachsten durch Reed-Kontakte realisiert. Der Schaltkontakt ist unter einem Schutzgas oder in einem Vakuum eingeschmolzen. Geschalten wird der Kontakt durch ein externes Magnetfeld. Dieser Gerätetyp wird häufig dazu verwendet um zu überprüfen, ob Fenster und Türen geschlossen sind. In Kombination mit Klimasensoren kann so zum Beispiel bei geöffnetem Dachfenster und Schlechtwetter eine Warnung ausgegeben oder gleich veranlasst werden, dass der entsprechende Aktor das Fenster schließt.

4.2 Aktoren

In der Steuer- und Regelungstechnik stellen Aktoren das Gegenstück zu Sensoren dar. Aktoren werden durch Sensoren getriggert und geben anschließend den Ausschlag zu einer mechanischen Arbeit bzw. Bewegung. In technischer Hinsicht sind Aktoren deutlich komplexer als Sensoren und daher auch teurer. Der entscheidende Faktor in der Größe des Netzwerks wird daher durch die Anzahl der Aktoren vorgegeben.

Die folgende Aufzählung stellt eine Liste typischer Aufgaben von Aktoren in Privathaushalten dar:

Funktion	**Einsatzgebiet**
Energiemanagement	Licht- und Klimasteuerung (z.B. Ventilsteuerung bei Radiatoren)
Zutrittskontrolle	sicherheitsbeschränkte Räume, automatische Rechneranmeldung
Gerätesteuerung	Automatische Türöffnung, Kaffeemaschine, Umschalten von TV auf Überwachungskamera bei Türklingeln, Wallscreens, Bewässerungsanlagen, Jalousienmotor, Türöffner

4.3 Übersicht

Die nachfolgende Tabelle stellt eine Auflistung von Sensoren dar, wie sie in einem Haushalt in naher Zukunft zu finden sein könnten. Dabei müssen nicht in jedem Raum alle Sensoren aus der Übersicht verwendet werden, die Auswahl wird vielmehr variieren. Die Summe der im Raum platzierten Geräte wird daher unter der Gesamtsumme dieser Aufzählung liegen. In der Spalte „Summe“ wird die Anzahl der Sensoren und Aktoren mit *S* bzw. *A* abgekürzt.

Sensortyp	**Anzahl Sensoren**	**Aktortyp**	**Anzahl Aktoren**	**Summe**
Bewegung und Licht	1-2 pro Raum	Rollladen-/ Markisensteuerung Lichtsteuerung Schalter für Endgeräte (Kaffeemaschine, Bewässerung, Wallscreens)	1 pro Gerät 1 pro Lampe 1 pro Endgerät	4 S 2 A 2 A 5 A
Temperatur	1 pro Raum	Thermostatregler Klimaanlagensteuerung	1 pro Radiator 1 pro Klimagerät	1 S 1 A 1 A
Druck	1 pro aktivem Sitzplatz	Schalter für Endgeräte (TV, PC, HiFi…)	1 pro Endgerät	3 S + 3 S
Feuer	1-3 pro Raum	Alarmgeber	1 pro Raum	3 S +1 A
Gas	1-2 pro Raum	Alarmgeber	1 pro Raum	2 S + 1 A
Wasser	1 pro Raum	Alarmgeber	1 pro Raum	2 S + 1 A
Glasbruch	1 pro Raum	Alarmgeber	1 pro Raum	1S + 1 A
Lesemodule	1 pro Tür	Elektrisches Türschloss	1 pro Tür	1 S + 1 A
Kontakt-Module	1 pro Haus	Alarmgeber bzw. elektr. Fenstermotor	1 pro Fenster	1 S + 2 A
Gesamtbedarf				**< 40 Geräte**

4.4 Fazit

In diesem Kapitel wurde ein Überblick gegeben, welche Typen von Sensoren derzeit verfügbar und einsetzbar sind. Dies bedeutet aber nicht, dass in jedem Raum eines typischen Haushalts, Sensoren jeder Klasse zu finden sind (ein Temperatursensor im Keller macht beispielsweise wenig Sinn). Vielmehr wird eine Obergrenze für ein technisch gut ausgestattetes Zimmer gegeben. Dieses Beispiel zeigt, dass der Bedarf von 40 Geräten nicht überschritten wird. Dies bedeutet, dass bei einer durchschnittlichen Familie die Anzahl von Geräten (abhängig von der Zimmeranzahl) maximal im Bereich um 350 liegen kann. Ein derzeitig hoher Preis für die Sensoren (Stand Frühjahr 2005: ca. 20 € / Stück) sowie die Notwendigkeit weiterer Geräte für die Ansteuerung von Geräten durch die Sensoren verhindern derzeit den Einsatz in diesem Umfang. Es ist daher realistisch, momentan von einer maximalen Netzwerkgröße mit weniger als 100 Geräten auszugehen. Da jedoch noch weitere Szenarien denkbar sind und der Preis in Zukunft wohl noch fallen wird, berücksichtigt diese Arbeit auch höhere Gerätezahlen.

5 Algorithmen

In diesem Kapitel wird aus verschiedenen Klassen von Pfadsuch-Algorithmen jeweils ein viel versprechender Algorithmus für die Verwendung in G-SPAN gewählt und genauer untersucht. Dabei handelt es sich um graphentheoretische, traditionelle und neuartige Suchverfahren. Dem Autor war bis zur Drucklegung keine Arbeit bekannt, die das Verhalten unterschiedlicher Klassen von Algorithmen für den Einsatz zur Pfadsuche vergleicht. Daher erläutert diese Arbeit in diesem Kapitel die grundlegende Funktionsweise anhand von mathematischen Definitionen und Beispielen in Pseudo-Code, bevor im nächsten Kapitel genauer auf die Implementierung eingegangen wird. Kleinere Beispiele helfen dem Leser, sich abschließend ein Bild über den Ablauf machen.

5.1 Maxflow

In der Klasse der graphentheoretischen Algorithmen, haben sich die Verfahren zur Lösung des maximalen Flusses (Maxflow) als besonders effizient erwiesen. Dies liegt zum Teil daran, dass das Problem schon „recht alt“ ist und die Verfahren daher recht ausgereift sind. Zu Beginn der 30er Jahre wurden in der damaligen Sowjetunion durch A.N. Tolstoi erste Verfahren [26;27] entwickelt um Güter auf schnellstem Wege über das Schienennetz zu transportieren. Nach dem Ende des 2. Weltkrieges wurde dieses Transportproblem dann erneut aufgegriffen, Hilfsgüter möglichst schnell an die Stellen zu bringen, die sie dringend benötigen. Dazu muss die zur Verfügung stehende Infrastruktur möglichst effizient genutzt werden. Diese Art von Problemen liegt in einem Grenzbereich zwischen der Informatik und der Unternehmensforschung (*Operations Research*). Derartige Probleme lassen sich durch eine Vielzahl komplexer mathematischer Gleichungen beschreiben und stellen ein lineares Optimierungsproblem dar. Dieses lässt sich auf klassische Weise mit der Simplex Methode lösen (siehe [28]). Da dieses Verfahren jedoch eine exponentielle Laufzeit aufweist, eignet es sich nicht für den Einsatz in größeren Netzwerken. Auf der Suche nach effizienteren Methoden sind dann beginnend mit dem Ford/Fulkerson Verfahren aus den 60er Jahren die Maxflow Algorithmen entstanden.

Zur Vereinfachung werden in diesem Beispiel ungerichtete Kanten verwendet. Die Klasse der Maxflow Algorithmen arbeitet nur auf gerichteten Graphen, weswegen implizit angenommen wird, dass alle Kanten von der Quelle in Richtung der Senke gerichtet sind. Die Knoten A und F repräsentieren dabei einzelne Arbeitsstationen, während die Kanten die Netzwerkverbindungen dieser Computer symbolisieren. Die übrigen Knoten B bis E stellen Router dar, über die die Netzwerkverbindungen geschaltet werden.

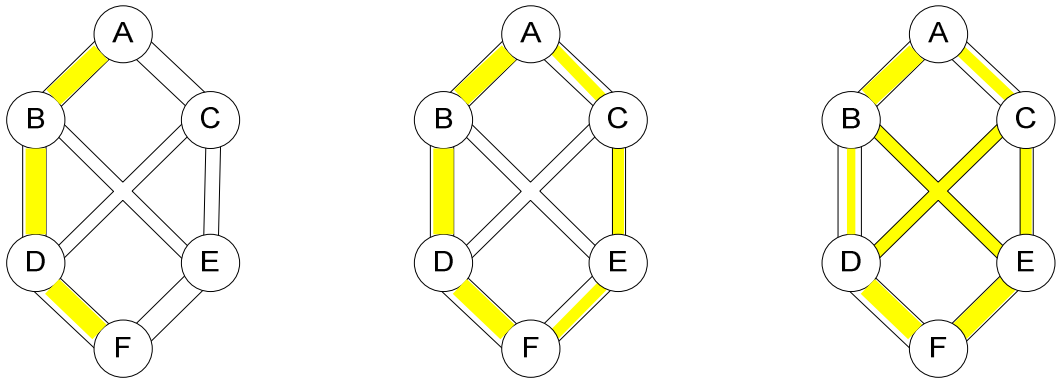


Abbildung 2 - Maximaler Fluss in einem Netzwerk

In diesem Beispiel soll ein Datenstrom von Server A möglichst schnell zu Zielrechner B übertragen werden. Die Leitungen über die Router (B bis E) besitzen eine feste Kapazität, die zu ihrer Größe proportional ist. Die Daten fließen dabei nur in Richtung des Ziels. An jedem Router wird entschieden, wie viele Daten über welche Leitung weiterverschickt werden. Dabei ist aber zu beachten, dass an den Zwischenknoten die Menge der eingehenden Daten der Menge der ausgehenden gleicht. Das Maximalflussproblem beschreibt die Aufgabe, die Verbindungen und ihre Kapazitäten so auszuwählen, dass der Fluss, der aus der Quelle austritt und an der Senke ankommt, maximal ist. Wird nur eine Datenleitung verwendet, so wird das Kommunikationsnetz nicht optimal ausgenutzt. Abbildung 2 zeigt, wie die Wahl der verwendeten Datenleitung den Durchsatz beeinflussen kann. Nehmen wir an, die Daten werden zuerst über die Leitung AB, BD und DF geschickt, wobei die Kapazitäten der ersten beiden Strecken vollständig, die der letzten beinahe vollständig ausgelastet wird (vgl. linke Skizze). Danach werden die Leitungen entlang ACEF gefüllt, bis die Kapazität von CE erschöpft ist. Die mittlere Skizze zeigt, dass der Durchsatz noch etwas weiter erhöht werden kann, wenn über ACDF noch weitere Daten geschickt werden. Diese Lösung ist jedoch nur suboptimal, wie die rechte Skizze zeigt. Durch Öffnen der Leitung BE, wird in DE eine ausreichend große Kapazität frei, die es erlaubt, die Leitung CD vollständig zu füllen.

Zur Darstellung des Problems wird das Netzwerk mit Hilfe von gerichteten Graphen modelliert. Dabei stellen die Knoten die einzelnen Geräte, die Kanten die Verbindungen und die Gewichte die Kapazitäten dar. Die mathematische Definition (vgl [29;30]) eines Flusses schildert sich wie folgt:

Ein Netzwerk $G=(V,E)$ ist ein gerichteter Graph mit $|V|=n$ und $|E|=m$. Jede Kante $(u,v)\in E$ besitzt eine nichtnegative Kapazität $c(u,v)\geq 0$. Für den Fall dass $(u,v)\notin E$, nehmen wir an, dass die Kapazität $c(u,v)=0$ ist. Im Graph gibt es zwei besondere Knoten: eine Quelle s und eine Senke t. Wir nehmen an, dass jeder weitere Knoten $v\in V$ auf einem Pfad von Quelle zu Senke liegt und somit ein Pfad $s\rightarrow v\rightarrow t$ existiert. Der zu Grunde liegende Graph ist daher zusammenhängend und es gilt: $|E|\geq V-1$.

Für Knoten $v\in V$ sei $E_{in}(v)=\{(u,v):(u,v)\in E\}$ die Menge der eingehenden Kanten und $E_{out}(v)=\{(v,u):(v,u)\in E\}$ entsprechend die Menge der abgehenden Kanten aus E. Ein Fluss stellt daher eine Funktion $f:E\rightarrow\mathbb{R}^{+}$ dar, die des Weiteren folgende Bedingungen erfüllt:

- Kapazitätsbeschränkung: $\forall\, e\in E: 0\leq f(e)\leq c(e)$

  Der Fluss einer Kante kann die gegebene Kapazität nicht überschreiten.

- Umgekehrte Symmetrie: $\forall\, u,v\in V: f(u,v)=-f(v,u)$

  Flüsse können negative Werte annehmen. Wird eine Kante in entgegen gesetzter Richtung durchlaufen, so gilt für sie dieselbe Kapazität, jedoch mit einem negativen Vorzeichen. Daraus folgt, dass der Fluss eines Knotens zu sich selber 0 ist, denn für alle $u\in V$ gilt $f(u,u)=-f(u,u)$.

- Flusserhaltung: $$0=\sum_{e\in E_{in}(v)} f(e)-\sum_{e\in E_{out}(v)} f(e)$$

  Diese Bedingung verdeutlicht, dass die Summe der in die Knoten hineinfließenden Flüsse gleich der Summe der aus den Knoten hinausfließenden Flüsse ist. Die Flusserhaltung besagt also, dass die Knoten genauso viel weiter geben wie sie erhalten.

Der Gesamtfluss $F = F(f)$ ist definiert durch

$$F = \sum_{e \in E_{in}(v)} f(Senke) - \sum_{e \in E_{out}(v)} f(Senke) \tag{0.1}$$

wobei zusätzlich gilt, dass der hintere Teil der Gleichung 0 ist, falls bei der Senke keine abgehenden Kanten existieren. Ein Fluss f auf einem Netzwerk N heißt Maximalfluss genau dann wenn

$$F(f) = \max\{F(f') : f' \; Fluss \; auf \; N\}.$$

Der Gesamtfluss wird durch zu Hilfenahme der Senke definiert. Es gibt jedoch eine effizientere Methode, die, wie später noch genauer erläutert wird, von den meisten Implementierungen verwendet wird. Diese berechnen den Gesamtfluss durch Schnitte. Den vollständigen Beweis erläutert [30]. Bei einem Schnitt wird der Graph in zwei Partitionen geteilt. Die Menge der Kanten, die die beiden entstandenen Knotenmengen miteinander verbindet, bildet den Schnitt. Mathematisch wird der Schnitt durch die Kantenmenge $(S;\overline{S}) \cup (\overline{S};S)$ definiert. Des Weiteren gilt:

$$\begin{aligned} &Es \; sei \; S \subseteq V \; mit \; q \in S, s \notin S \; und \; \overline{S} = V \backslash S. \\ &(S;\overline{S}) = \{(u,v) : (u,v) \in E \wedge u \in S \wedge v \in \overline{S}\}, \\ &(\overline{S},S) = \{(u,v) : (u,v) \in E \wedge u \in \overline{S} \wedge v \in S\}. \end{aligned}$$

Die Berechnung des Flusses kann daraufhin durch jeden Schnitt erfolgen. Dazu wird die Kapazität auf einem Schnitt definiert durch die Summe der Kapazitäten der Kanten, die den Schnitt bilden, also aller Kanten zwischen den beiden Knotenmengen: $c(S) = \sum c(e) \mid e \in (S;\overline{S})$.

Dadurch gilt offenbar, dass der Fluss maximal so groß werden kann wie die Kapazität des Schnitts:

$$\begin{aligned} &\forall \; S \subseteq V \; mit \; q \in S, s \notin S \; gilt \\ &\sum_{e \in (S;\overline{S})} f(e) - \sum_{e \in (\overline{S};S)} f(e) = F \leq c(S) \end{aligned}$$

Für den Fall $F = c(S)$ gilt, so ist der Fluss im Netzwerk maximal und die Kapazität des Schnitts minimal. Viele Algorithmen basieren auf dieser Gleichheit, die daher auch unter dem *max-flow-min-cut* Theorem (maximaler Fluss / minimaler Schnitt) bekannt ist. Die Definition des maximalen Flusses lautet demnach:

$$\max\{F(f) : f \; Fluss \; auf \; N\} \leq \min\{c(S) : S \subseteq V, q \in S, s \notin S\}$$

5.1.1 Die Ford/Fulkerson Methode

Wie bereits erwähnt, war die Ford/Fulkerson Methode eine der ersten, die für das Problem des maximalen Flusses eine Lösung bereitstellte. Ihre Methode gewährt einen Spielraum bei der Wahl der verwendeten Heuristik (vgl. [31]), weshalb mittlerweile viele verschiedene Implementierungen mit unterschiedlichen Laufzeiten existieren. Die grundlegenden Elemente sind jedoch überall gleich: residuale Netze, flussvergrößernde (augmentierende) Pfade und Schnitte.

Mit einem flussvergrößernden Pfad bezeichnet man einen Weg von Quelle zu Senke, dessen Kapazitäten noch nicht voll erschöpft sind. Die Summe dieser Pfade bildet das Residualnetzwerk (siehe weiter unten im Text). Wird in diesem kein weiterer Pfad gefunden, so hält der Algorithmus.

Die Ford/Fulkerson Methode arbeitet iterativ. Zu Beginn wird der Fluss über jede Kante auf 0 gesetzt. Mit jedem Iterationsschritt wird der Fluss entlang eines augmentierenden Pfades erhöht. Dies geschieht so lange, bis keine solchen Pfade mehr gefunden werden können. Der Fluss, der bei der Terminierung entsteht, entspricht einem maximalen Fluss.

```
Ford-Fulkerson-Methode (G, q, s)
1     ∀u,v ∈ V : f(u,v) = f(v,u) = 0  (Nullfluss)
2     while es existiert ein augmentierender Pfad p im Residualgraphen
3         do c_res(p) = min{c_res(u,v) : u,v ∈ p}
4         for ∀(u,v) : (u,v) ∈ p
5                 do      f(u,v) = f(u,v) + c_res(p)
6     .                   f(v,u) = −f(u,v)
7     return f
```

Wird dieses Verfahren auf das Beispiel aus Abbildung 2 angewendet, so ergibt sich folgende neue Konstellation:

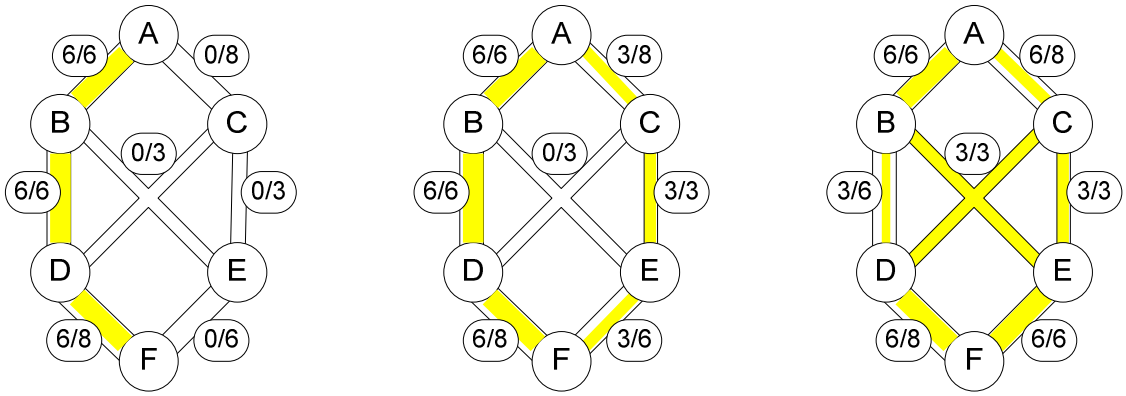


Abbildung 3 - Bestimmung des maximalen Flusses

Die Ford/Fulkerson Methode sieht vor, den Fluss entlang eines Pfades zu erhöhen, so lange es noch ungenutzte Kapazitäten entlang aller Kanten eines Pfades gibt. In der linken Skizze geschieht dies entlang ABDF und in der mittleren Skizze dann anschließend entlang ACEF. Außer dieser trivialen Art das Netz zu füllen, gibt es noch eine weitere Methode. Betrachtet man willkürliche Pfade im Netzwerk und nicht nur solche von Quelle zu Senke längs des Pfades, so ergeben sich weitere Möglichkeiten. Dazu wird eine weitere Kenngröße eingeführt: die *Residualkapazität* c_{res} einer Kante e beziehungsweise eines Netzwerks:

$$Kante: \forall e \in E: c_{res}(e) = c(e) - f(e)$$
$$Netzwerk: \forall e \in E: E_{res} = \{e: c_{res}(e) > 0\}$$

Sie ist definiert durch die Differenz der Kapazität und des momentanen Flusses über diese Kante. Gleichzeitig stellt sie damit das Gewicht der Kante dar, wenn sie in umgekehrter Reihenfolge (also von der Senke zur Quelle längs des Pfades) durchlaufen wird. Eine Vergrößerung des Flusses kann somit stattfinden, in dem der Fluss, in von der Quelle zur Senke zeigenden Kanten (so genannte *Vorwärtskanten*), erhöht wird und in den von der Senke zur Quelle zeigenden Kanten (*Rückwärtskanten*) um denselben Betrag erniedrigt wird. Dadurch wird erkannt, dass der Fluss in obigem Beispiel entlang des Pfades ACDBEF weiter erhöht werden kann. Die Teilstrecken AC und CD werden um drei Einheiten erhöht, während drei Einheiten von BD umgelenkt werden und nun über BE und EF fließen. Der Fluss in DF ändert sich nicht, da die Einheiten, die nun von BD weniger ankommen, durch den Strom über CD ausgeglichen werden. Der Gesamtfluss kann daher in der rechten Skizze im Vergleich zur mittleren von neun Einheiten auf zwölf Einheiten gesteigert werden.

Es zeigt sich, dass die Möglichkeiten zur Flusserhöhung durch das Minimum der Flüsse in den Rückwärtskanten (entspricht der Residualkapazität) begrenzt sind. Folglich terminiert die Methode genau dann, wenn es auf jedem Pfad nur noch volle Vorwärts- und leere Rückwärtskanten gibt und liefert damit den maximalen Fluss.

5.1.2 Laufzeit

Die Laufzeit der Ford/Fulkerson Methode hängt davon ab, wie der augmentierende Pfad (Zeile 3 im Algorithmus) gefunden wird. Bei einer ungünstigen Wahl kann der Fall auftreten, dass der Algorithmus nicht terminiert. Wird die Breitensuche verwendet, um augmentierende Pfade zu finden, läuft der Algorithmus in polynomialer Zeit. Für die Laufzeitbestimmung der allgemeinen Methode wird davon ausgegangen, dass die Pfade willkürlich gewählt werden und dass es sich bei den Kapazitäten nur um ganzzahlige Werte handelt. Der Nullfluss wird in $O(m)$ Schritten erreicht. Die **while** Schleife (Zeile 2 im Code) wird in jedem Schritt, um mindestens „1“ erhöht. Dadurch läuft sie maximal $|f|$ mal ($|f|$ stellt dabei den Maximalfluss dar).

Durch geeignete Datenstrukturen kann der Residualgraph effizient gespeichert werden: $G_{res} = (V, E')$ mit $E' = \{(u,v) : (u,v) \in E \vee (v,u) \in E\}$. Kanten im Netzwerk G sind somit auch Kanten in G_{res}. Somit können Kapazitäten und Flüsse einfach in dieser Struktur gespeichert werden. Für einen Fluss f auf G, besteht der Residualgraph G_{res} aus allen Kanten $(u,v) \in G' : c(u,v) - f(u,v) \neq 0$. Die Zeit, die benötigt wird um im Residualgraph einen Pfad zu finden, ist daher $O(m') = O(m)$, sofern Tiefen- oder Breitensuche verwendet wird. Da jede Iteration der **while** Schleife $O(m)$ Schritte benötigt, beträgt die Gesamtlaufzeit der Ford/Fulkerson Methode $O(m|f|)$.

Wie der augmentierende Pfad gefunden wird, wird durch die Methode allerdings nicht definiert, weshalb die Obergrenze nur theoretisch gilt. Dabei gibt es mehrere Heuristiken, wie dieser Pfad gewählt wird.

- Kürzeste Pfade zuerst (*shortest path first*) [32]

  In diesem Fall werden die Pfade mit einer minimalen Kantenanzahl von Quelle zu Senke zuerst gewählt. Die Laufzeit ist dadurch unabhängig von der Größe des Flusses und es ergibt sich eine Verbesserung der Laufzeit auf $O(nm^2)$.

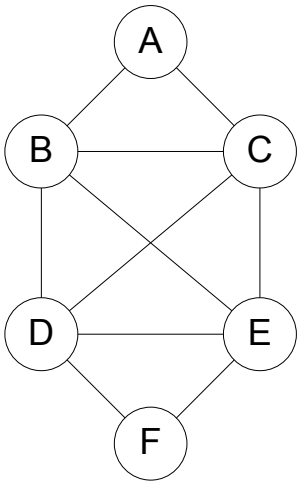
- Push – Relabel [33]

  Diese Klasse von Algorithmen verletzt absichtlich temporär die Flussbedingung, d.h. dass die Summe der eingehenden Flüsse nicht gleich der Summe der ausgehenden Flüsse ist. Durch dieses Verhalten können Flüsse parallel (anstatt bisher nur sequentiell) gesucht werden. Unter Verwendung geeigneter Heuristiken für Push-Relabel Algorithmen (*Global & Gap Relabeling*) erreichen diese eine Laufzeit von $O(nm \log {n^2}/{m})$

Obwohl in Theorie und Praxis (siehe dazu [16;33-41]) nachgewiesen werden konnte, dass neuere Maxflow Algorithmen (z.B. Push-Relabel Algorithmen) im allgemeinen kürzere Laufzeiten aufweisen als Algorithmen mit augmentierender Pfaden, so gilt dies nicht unbedingt für die Pfadsuche in s-verbundenen Graphen in G-SPAN. Hierbei handelt es sich um einen Graphen, dessen Kanten alle eine Kapazität von „1“ aufweisen. Die Anzahl der Schritte, in denen augmentiert werden muss ist daher minimal und der Vorteil neuerer Verfahren fällt damit für die hier verwendete Graphstruktur weg (siehe [34]). In G-SPAN wurde daher ein an Ford/Fulkerson angelehnter Algorithmus unter Verwendung der Breitensuche zur Pfadsuche herangezogen.

5.2 Die Tiefensuche

Einer der bekanntesten klassischen Suchalgorithmen ist die Tiefensuche und wurde zunächst primär dazu verwendet, Graphen auf Zyklen zu untersuchen. Dadurch stellt sie aber auch einen natürlichen Weg dar, alle Knoten in einem Graph systematisch zu besuchen. Ausgehend vom Startknoten wandert das Verfahren zuerst in die „Tiefe“ und dann in die „Breite“. Das heißt, man wandert von ausgehenden Knoten zu einem noch nicht besuchten Nachbarknoten und führt den Algorithmus dort fort. Es werden keine Einschränkungen über die Ordnung festgelegt. Bei der *pre-order* Variante werden die besuchten Knoten beim erstmaligen Besuchen gelistet, bei der post-order Variante werden die Knoten erst gelistet, wenn sie keine weiteren nicht besuchten Nachbarn aufweisen. Daher können verschiedene Implementierungen unterschiedliche Ergebnisse liefern. Die folgende Abbildung verdeutlicht das Resultat der Tiefensuche mit *pre-order* an einem Beispielgraphen.



Tiefensuche: A B D F E C

Abbildung 4 - Die Tiefensuche

Sei $G = (V, E)$ ein Graph mit der Knotenmenge $V = \{1, 2, ..., n\}$ der Kantenmenge E, sowie der Menge der Nachbarknoten $N(v)$. Der Tiefensuchalgorithmus (vgl. [42]) gliedert sich wie folgt:

Die Tiefensuche (v)

```
var        besucht : array [1..n] of Boolean;
           v : 1..n;
procedure      tiefensuche (v : 1..n);
var:           w : 1..n;
begin
    besucht[v] := true; write(v);
    for all w ∈ N(v) do
            if not besucht[w] then tiefensuche(w)
end;
begin
    for v := 1 to n do
            besucht[v] := false;
    while ∃ v : besucht[v] = false
    do tiefensuche(v)
end.
```

Unter Verwendung von Adjazenzlisten zur Implementierung hat der Algorithmus eine Laufzeit von $O(|V|+|E|)$. Weitergehende Informationen zur Tiefensuche finden sich in der Standardliteratur.

5.3 Rekursiver Schlüsselaufbau (RKEP)

Bei RKEP handelt es sich um ein Verfahren, welches an der Universität Stuttgart entwickelt wird. Eine vollständige Beschreibung des Algorithmus wird in [43] erscheinen.

RKEP findet knotendisjunkte Pfade durch Fluten des Netzwerks mit speziellen Anfrage Paketen (*Queries*). Nachdem der Startknoten die *Query* losgeschickt hat, wird diese von den Knoten rekursiv an alle weiteren Schlüsselpartner weitergeleitet, (Fluten) bis ein Knoten meldet, dass er das Ziel kennt. Dieser sendet dann eine Erfolgs-Nachricht (*Establish*) an die Knoten zurück, von denen er eine Anfrage erhalten hat. Jeder Knoten, der vormals eine *Query* Nachricht geschickt hat, erhält beim rückwärtigen Fluten der *Establish* Nachrichten unter einer gewissen Voraussetzung an die Graphstruktur (siehe weiter unten) insgesamt bis zu z dieser Nachrichten. Damit ist es dem Knoten möglich, über die Knoten von denen er diese Nachricht erhält einen Schlüssel zum gesuchten Knoten aufzubauen. Daraufhin schickt dieser Knoten eine *Establish* Nachricht an die Knoten weiter, die ihm vormals eine *Query* geschickt haben (siehe Abbildung 5).

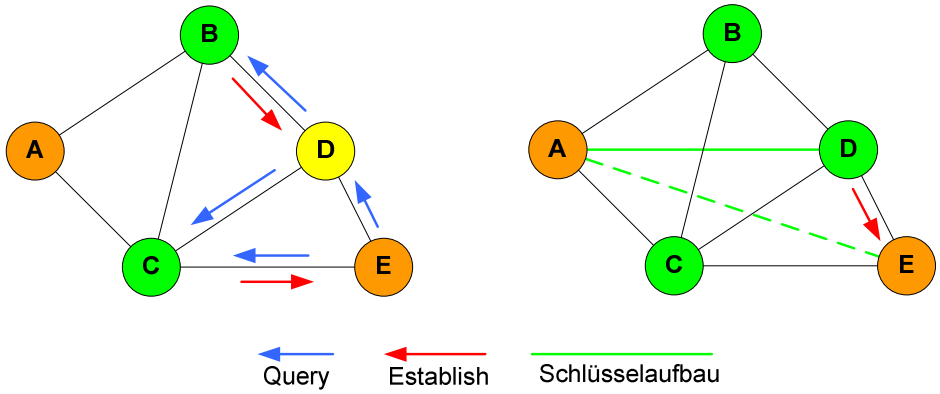


Abbildung 5 - Paketverhalten in RKEP (s=2)

Das Verfahren endet, wenn der initiierende Knoten s *Establish* Nachrichten erhalten hat. In diesem Fall baut er einen Schlüssel zum Zielknoten auf und das Ziel ist erreicht. Bei RKEP wird daher weniger von einem Pfadsuch-Algorithmus, sondern vielmehr von einem Schlüsselaufbau-Algorithmus gesprochen.

Um die Funktionalität von RKEP zu garantieren, werden gewisse Anforderungen an den Graphaufbau und somit auch an die Art und Weise, welches Knotenpaar den initialen Schlüsselaufbau durchführt, gestellt. Wird auf diese Forderung verzichtet, so kann es vorkommen, dass einige Knoten nicht genug *Establish* Nachrichten erhalten und der Algorithmus in einem undefinierten Zustand endet. In der nachfolgenden Abbildung wird dies verdeutlicht. Knoten A startet den Schlüsselaufbau zu Knoten J, indem er *Query* Pakete an seine Nachbarn versendet.

Diese starten ihrerseits auch einen Schlüsselaufbau zu Knoten J. Knoten I und G melden an ihre Nachbarknoten ein *Establish*. Knoten H empfängt $2 = z$ *Establish* Nachrichten und kann somit auch einen Schlüssel zu J aufbauen. Anschließend verschickt er eine *Establish* Nachricht an seine Nachbarn. Durch den Graphaufbau empfängt jedoch kein anderer Knoten ausreichend *Establish* Nachrichten, um einen weiteren Schlüssel aufzubauen. Der Algorithmus endet ohne erfolgreichen Schlüsselaufbau.

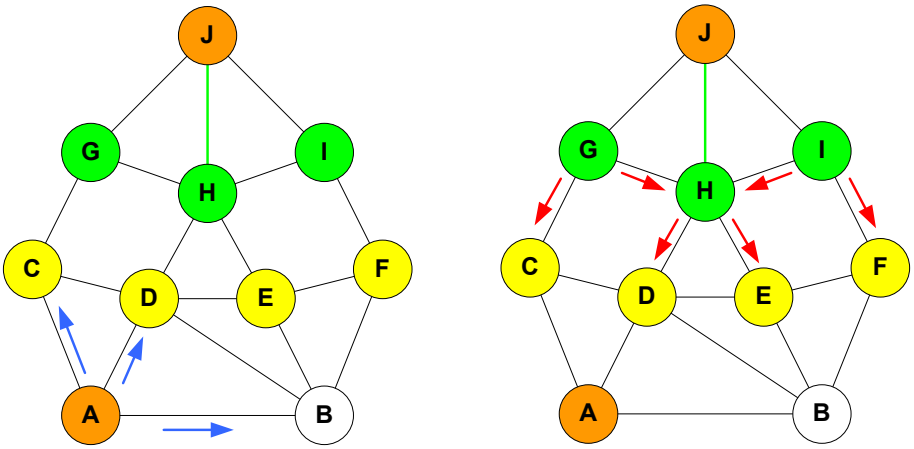


Abbildung 6 - fehlerhafter Graphaufbau für RKEP

Wird hingegen während des Graphaufbaus der hinzugefügte Knoten immer an z benachbarte Knoten imprintet, tritt dieser Fall nicht auf. Die Knoten an denen ein neuer Knoten imprintet wird, bilden einen so genannten s-Connector. Dieser garantiert, dass RKEP terminiert. Veranschaulicht werden kann dies am besten am Besten an einem Beispielgraphen für $z = 2$ (siehe Abbildung 7), da in diesem Fall die Knoten mit ihren Verbindungen lauter Dreiecksformen bilden. Über den s-Connector kann jede *Establish* Nachricht zu einem Knoten weitergeleitet werden. Dies garantiert, dass ein Knoten die Nachricht immer weiterleiten kann und damit auch den Knoten, der die Anfrage initiiert hat, erreicht. Für $z = 3$ kann man sich den s-Connector als Tetraeder vorstellen, größere Werte jedoch sind multi-dimensional, dass sie nur noch schwer vorstellbar sind.

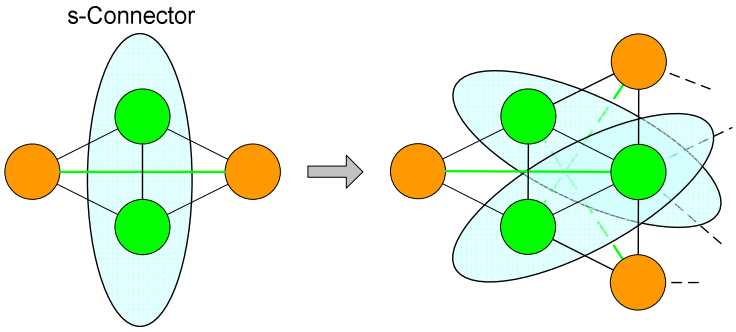


Abbildung 7 - RKEP s – Connector

Um die benötigte Graphstruktur garantieren zu können, stehen zwei verschiedene Methoden zur Auswahl:

- Imprinten via Steuergerät

  Die Intelligenz zum korrekten Imprinten kann zum einen auf das Master Device ausgelagert werden. Dieses wählt die Schlüsselpartner für jeden neuen Knoten dann korrekt aus.

- Imprinten mit automatischer Korrektur

  Zum anderen kann die Aufgabe durch das Versenden von Nachbar-Paketen (*Neighborlists*) auch automatisch gelöst werden. Diese beiden Varianten werden in Kapitel 6.6 bzw. 6.7 weiterführend beschrieben.

6 Implementierung

Dieses Kapitel erläutert die Implementierung der Algorithmen. Dabei werden die verschiedenen Varianten bezüglich ihrer Funktionsweise und der Änderungen beziehungsweise Erweiterungen gegenüber den Standardverfahren (siehe vorangegangenes Kapitel) näher beschrieben. Die anschließende Analyse verschiedener Faktoren stellen Grenzen dar, die Implementierungen theoretisch nicht überschreiten dürfen. Dies wird im Kapitel Evaluation überprüft. Konkret handelt es sich dabei um folgende Varianten:

- Maxflow Algorithmus
  - basierend auf *Link State* Paketen (periodisches Versenden der Nachbarn)
  - basierend auf *Global View* Paketen (einmaliges Versenden der Datenbank)
- Tiefensuche
- RKEP
  - mit manuellem Imprinten (Netzwerkaufbau durch externe Geräte)
  - mit automatischem Imprinten (selbstkorrigierenden Netzwerkaufbau)

Im Anschluss werden die Basis–Funktionen der Knoten im Netzwerk kurz erläutert. Dabei handelt es sich im Wesentlichen um Anforderungen an die Software und nicht an die Hardware. Eine umfangreichere und detaillierte Beschreibung dafür wird in [11] und [12] gegeben. Da die Basisfunktionen vom gewählten Algorithmus unabhängig sind, werden sie in den Messungen nicht berücksichtigt.

Um die theoretisch maximale Paketanzahl zu bestimmen, sind einige Betrachtungen in z-verbundenen Graphen notwendig. Dazu gehört die Bestimmung der *maximalen Kantenanzahl*. Für die ersten z Knoten tritt hier eine Sonderbehandlung ein, da der Graph in diesem Fall voll vermascht ist. Danach werden mit jedem weiteren Knoten z weitere Kanten hinzugefügt, so dass sich die Gesamtzahl folgendermaßen bestimmen lässt:

$$m = (n - z) * z + \frac{z(z-1)}{2}$$

Ein weiterer Wert von großer Wichtigkeit ist die *maximale Entfernung* zweier Knoten im Netzwerk. Damit wird die **minimale** Anzahl von Knoten, die sich auf einem Pfad vom Start- bis hin zum Zielknoten befindet, bezeichnet. Dieser Wert kann bei einigen Protokollen (vgl. Maxflow 6.3.2) dazu verwendet werden, die Zahl der Nachrichten für die Pfadsuche nach oben abzuschätzen. Dazu wird von einer kettenförmigen Graphkonstellation ausgegangen, da hier die maximale Entfernung am größten ist. Bei jeder anderen Anordnung nimmt die maximale Entfernung ab.

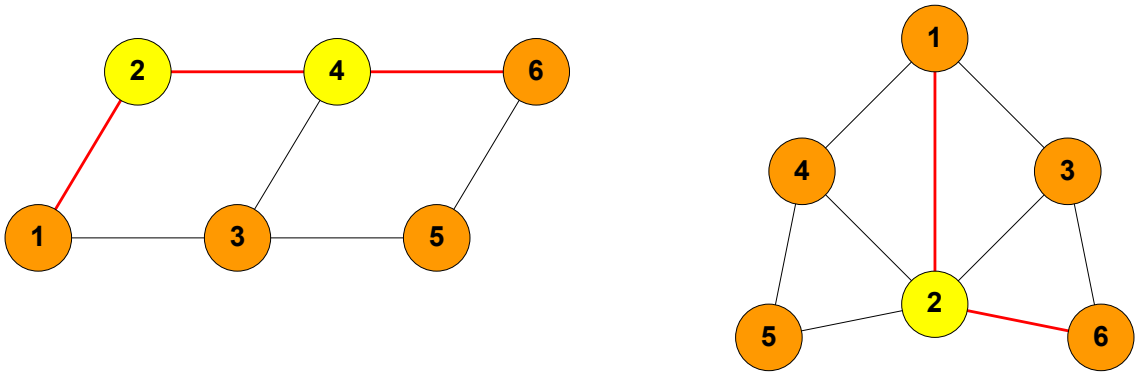


Abbildung 8 - maximale Entfernung (links: kettenförmig, rechts: zufällig)

Der linke Graph aus Abbildung 8 zeigt die maximale Entfernung im Graphen von Knoten $1 \rightarrow 6$ mit dem Abstand 2. Im rechten Graphen ist die maximale Entfernung kleiner, da sich von jedem beliebigen Startknoten jeder andere Knoten über nur einen weiteren Knoten erreichen lässt.

Die Abschätzungen für die theoretische Obergrenze der Paketanzahl sowie für den benötigten Speicher werden so exakt wie möglich angegeben. Um besser zwischen den Protokollen zu vergleichen, werden diese Werte für $z = 2$ weiter vereinfacht.

6.1 Kritische Erfolgsfaktoren

Anhand von Erfolgsfaktoren lassen sich verschiedene Algorithmen untersuchen und gegeneinander vergleichen. In G-SPAN basiert die Kommunikation auf einem Funknetzwerk, also einem gemeinsamen genutzten Medium (*shared Media*). Daher ist in der Analyse der Protokolle die Anzahl der Pakete, die die Auslastung des Netzwerks darstellen, eine entscheidende Größe. Da in G-SPAN nicht alle Pakete dieselbe Paketgröße aufweisen, ist es zudem wichtig zu überprüfen wie viele Daten insgesamt verschickt werden.

Nachrichtenanzahl:

Unter der Nachrichtenanzahl werden alle Pakete zusammengefasst, die über das Netzwerk geschickt werden. Eine von einem Knoten geschickte Nachricht, die von einem weiteren Knoten weitergeleitet wird, wird daher auch mehrmals gezählt. Es wird zwischen zwei Klassen von Nachrichten unterschieden, die je nach Typ des Algorithmus (lokal oder verteilt) in ihrer Anzahl sehr stark divergieren.

Unter *Setup* Nachrichten werden alle Nachrichten verstanden, die während des Aufbaus des Netzwerks verschickt werden. Also alle Nachrichten, die ab dem Hinzufügen des ersten bis hin zum letzten Sensor untereinander ausgetauscht werden.
Im Gegensatz dazu werden *Pathsearch* Nachrichten erst zu Beginn der Pfadsuche verschickt, die ein bestehendes Netz voraussetzt. Mit dem Erhalt einer Bestätigungsnachricht über den erfolgreichen Schlüsselaufbau am Startknoten wird die Zählung beendet.

Paketlänge:

Die Paketlänge hat einen maßgeblichen Einfluss darauf wie viele Daten über das Netzwerk geschickt werden. Eine große Anzahl kleiner Pakete kann dieselbe Netzwerkauslastung wie eine geringere Anzahl größerer Pakete verursachen. Bei der nachfolgenden Analyse der Protokolle wird daher unter Berücksichtigung der Anzahl der Pakete auch ihre Länge untersucht.

Robustheit:

Durch die Verwendung von UDP Paketen zur Kommunikation zwischen den Sensoren ist es möglich, dass nicht alle Pakete den jeweiligen Zielknoten erreichen, sondern schon vorher verworfen werden. Ein wichtiges Kriterium ist daher die Fehlertoleranz des eingesetzten Protokolls. Wird davon ausgegangen, dass keine Informationen verloren gehen, so kann dies im Fehlerfall zu einem nicht definierten Zustand führen. Dies wiederum führt zu Inkonsistenzen im Netzwerk und im ungünstigsten Fall dazu, dass der Graph durch den Ausfall eines Knotens nicht mehr z-verbunden ist.

Pfadlänge:

Unter der Pfadlänge wird die Anzahl der Sensoren (*Hops*) verstanden, die einen Teilschlüssel (*Keyshare*) weiterleiten. Start- und Zielknoten werden hierbei nicht berücksichtigt. Die restliche Knotenanzahl wird dann durch die Anzahl der disjunkten Pfade geteilt und ergibt den Mittelwert für die Pfadlänge:

$$Pfadlänge = \frac{weiterleitende\ Sensoren}{z}$$

Dieser Wert sowie die Gesamtnachrichtenzahl beeinflussen mehrere wichtige Aspekte während der Betriebszeit der Sensoren:

- In einem Ad-Hoc Netzwerk kommunizieren unabhängige Teilnehmer miteinander, d. h. dass nicht alle Sensoren dauerhaft miteinander Pakete austauschen. Nicht aktive Knoten können somit in einen Zustand zurückfallen, der energieeffizient ist. Erhöht sich die Pfadlänge, so müssen mehr Knoten aus diesem Zustand geholt und aktiviert werden. Dies wirkt sich nachteilig auf die Gesamtkapazität des Netzwerks aus. Dieses kann dadurch zwar nicht kompromittiert werden, jedoch kann der Betrieb eingeschränkt werden (*Battery Draining Attack* [44]). Dieser Effekt ist nicht nur auf die Pfadlänge begrenzt. Eine Erhöhung der Gesamtnachrichtenanzahl im Netzwerk wirkt sich sogar noch deutlich stärker aus. Fallen Knoten aufgrund eines zu geringen Energieniveaus aus, kann es zu **Netzwerkpartitionen** kommen, da die Entfernung zwischen Sensoren die maximale Sendereichweite übersteigt.
- Des Weiteren wirkt sich die Pfadlänge auch unmittelbar auf die Sicherheit im Netzwerk aus. Je länger die Pfade sind, desto mehr Knoten übermitteln Schlüssel. Dies kann es einem Angreifer erleichtern **Informationen abzufangen**. Denn die Wahrscheinlichkeit, einen am Schlüsselaufbau beteiligten Sensor aus der Menge aller Geräte herauszufinden, steigt.
- Ein erhöhtes Paketaufkommen erfordert höhere Bandbreiten und führt somit zu technisch aufwendigeren und komplexeren Geräten. Dies wirkt sich unmittelbar in einer Erhöhung der **Produktionskosten** aus (vgl. [45]).

6.2 Basis–Funktionen

Zum Aufbau des Netzwerks ist es notwendig, zwischen den Geräten einen Schlüssel aufzubauen um eine sichere Kommunikation zu ermöglichen. Dies geschieht in der Regel über das Imprinting unter der Annahme, dass ein sicherer Kanal zur Verfügung steht. Alle Knoten befinden sich nach ihrer Startsequenz im Zustand „an“ (*up*) und haben keinen Schlüssel in ihrem Schlüsselspeicher. Mit jedem Imprinting wird nun ein Schlüssel hinzugefügt. Beim Speichern des *z-ten* Schlüssels wechselt der Zustand auf „bereit“ (*ready*). Ab diesem Zeitpunkt hat der Knoten genug Schlüssel, um über z knotendisjunkte Pfade mittels Pfadsuche einen Schlüssel zu jedem anderen Knoten im Netzwerk aufzubauen.

Eine weitere globale Funktion ist die Schlüsselerstellung. Diese wird nicht nur – wie eben beschrieben – beim Imprinting verwendet, sondern auch bei der Pfadsuche. Sind alle benötigten knotendisjunkten Pfade gefunden, so wird nacheinander über jeden Pfad ein Teilschlüssel geschickt. Jeder dieser Teilschlüssel wird unter Verwendung der XOR-Funktion mit den vorhergehenden Teilschlüsseln im Schlüsselspeicher abgelegt. So wird sichergestellt, dass nur der Sender und der Empfänger über den vollständigen Schlüssel verfügen, sofern ein Angreifer nicht mehr als s Knoten (für jeden knotendisjunkten Pfad einen) übernommen hat.

6.3 Maxflow (LSP)

In dieser Variante wird der Maxflow Algorithmus nach Ford/Fulkerson zur Suche nach knotendisjunkten Pfaden implementiert. Dieser läuft vollständig lokal auf dem Knoten ab. Das bedeutet, dass bis zum Verschicken der Teilschlüssel keine Kommunikation mit weiteren Knoten notwendig ist. Gleichzeitig bedeutet es aber auch, dass alle Knoten während der Setupphase mehr Nachrichten verschicken, als ein verteilter Algorithmus. In dieser Variante baut jeder Knoten lokal eine globale Sicht des Netzwerks auf. Dies wird durch das Versenden von Nachbarlisten, so genannter *Link State* Pakete (LSP), erreicht.

Die *Link State* Pakete werden gemäß RFC1583 (vgl. [46]) periodisch (in dieser Variante ca. alle 8 Sekunden) an alle Nachbarknoten verschickt. Jeder Knoten aktualisiert beim Empfang dieser Pakete seine lokale Datenbank mit den Informationen des Paketes und leitet diese Informationen an seine Nachbarn weiter. Durch das Fluten der *LSP*s im Netzwerk erhalten alle Knoten die Informationen jedes Knotens. Dies hat ein sehr großes Nachrichtenaufkommen zur Folge.

In einem Distanz-Vektor Routing Protokoll tritt dieser Fall nicht ein. Dieses hat aber den Nachteil, dass der Ausfall eines Knotens nicht korrekt abgefangen wird. Es kommt zum „*count to infinity*“ Problem (siehe [47]). Um den Nachrichtenaufwand unter der Verwendung der Link State Pakete zu minimieren, haben diese einen speziellen Aufbau.

Für den Einsatz in G-SPAN werden jedoch nicht alle Einträge aus der Definition im RFC benötigt und können eingespart werden. Die vereinfachte Form der *LSP*s sieht folgendermaßen aus:

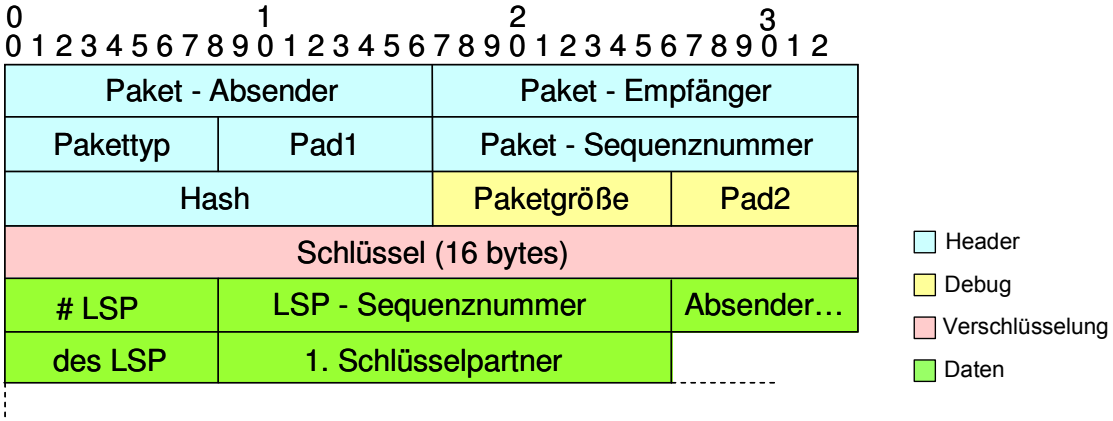


Abbildung 9 - Aufbau eines Link State Paketes für das Maxflow Protokoll

- #LSP

  Dies steht für die Anzahl der Einträge im LSP. Der Wert beträgt mindestens z (für die Anzahl initialer Schlüsselverbindungen) und ist nach oben nur durch die maximale Paketlänge (siehe weiter unten) begrenzt.

- LSP – Sequenznummer

  Jedes LSP verfügt über eine eigene Sequenznummer. Dies ist notwendig um Veränderungen im Netzwerk festzustellen. Da es keine global eindeutige Instanz im Netzwerk gibt, die streng monoton wachsende Sequenznummern vergeben kann, muss jeder Knoten diese Aufgabe selbst übernehmen. Dies wirft zugleich ein weiteres Problem auf. Nun kann ein Knoten mehrere Pakete mit derselben Sequenznummer erhalten. Gelöst wird dieses Problem durch die Zusammensetzung des Primärschlüssels aus der Sequenznummer und dem Absender des Pakets.

- Der Absender des LSPs

  Beim Weiterleiten jedes Pakets trägt der Knoten beim Absenden seine Knoten ID in das Feld „Paket-Absender“ ein. Um noch feststellen zu können welcher Knoten das LSP erstellt hat ist ein zweites Feld notwendig. Dessen Knoten ID wird daher im Feld „Absender des LSP“ hinterlegt.

- Die Schlüsselpartner

  In diesem und den folgenden Feldern werden alle Schlüsselpartner des Absenders des LSPs gelistet. Nach dem automatischen Imprintvorgang beträgt die Anzahl mindestens z. Bei einem erneuten Versenden von *Link State* Paketen können später erstellte Schlüssel berücksichtigt werden. Dies ist jedoch nur so weit möglich wie es die maximale Paketlänge zulässt:

$$Anzahl\ Schlüssel = \frac{Paketgröße - Headergröße - 1}{2}\ (in\ Bytes)$$

  Damit sich das Wissen über die Schlüsselpartner auf alle Knoten im Netzwerk verteilt, müssen die *Link State* Pakete weitergeleitet werden. Dazu versendet jeder Knoten alle „neuen“ LSPs an seine Schlüsselpartner, mit Ausnahme desjenigen, von dem er das Paket empfangen hat, weiter.

6.3.1 Setup

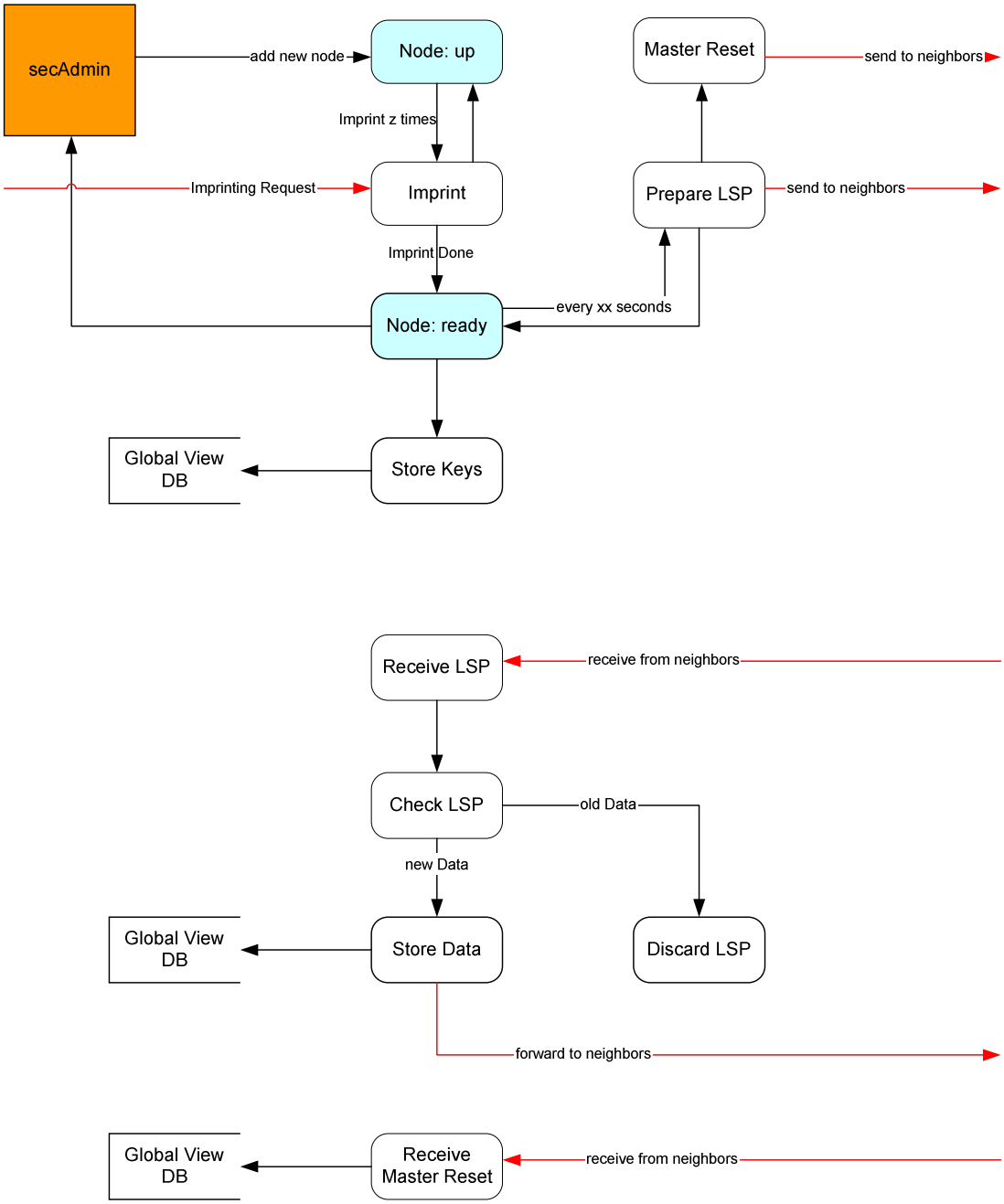


Abbildung 10 - Aufbau des Netzwerks unter Maxflow (LSP)

Das Administrationswerkzeug secAdmin initiiert die Knoten mit dem Zustand *up* und imprintet sie mit z weiteren Knoten. Daraufhin wechseln die Knoten in den Zustand *ready*, speichern die Informationen zu den Schlüsselpartnern in ihrer lokalen Datenbank (*Global View*) und beginnen mit ihrer Senderoutine.

- Verschicken der Schlüsselinformationen (*Prepare LSP*)

  Nachdem der Knoten genug Schlüssel im Schlüsselspeicher abgelegt hat, (mindestens z) beginnt er *Link State* Pakete zu generieren und diese an all seine Schlüsselpartner im Netzwerk zu verschicken. Dies geschieht periodisch nach einer frei konfigurierbaren Zeitspanne in Abhängigkeit mit der Anzahl und Mobilität eventuell vorhandener mobiler Geräte. Um auf Veränderungen im Netzwerk reagieren zu können, wird die Sequenznummer in jeder Periode erhöht.

- Rücksetzen der Datenbank (*Master Reset*)

  Je kürzer die Zeitspanne zwischen zwei Perioden beim Versand von LSPs ist, desto aktueller ist die Sicht der Knoten auf das Netzwerk. Gleichzeitig steigt die Sequenznummer aber auch schneller an und der Überlauf muss abgefangen werden. Dazu verschickt der Knoten ein *Master Reset* Paket an seine Schlüsselpartner.

Empfangsroutine

- Empfang von Link State Paketen (*Receive*)

  Um unnötige Schreiboperationen zu verhindern, muss zunächst überprüft werden, ob es sich bei dem LSP nicht um ein Duplikat handelt, dessen Informationen schon in der Datenbank gespeichert sind. Wie schon in Kapitel 6.3 beschrieben, genügt es nicht, die Pakete nur aufgrund ihrer Sequenznummer mit der lokalen Datenbank zu vergleichen, da es in einem asynchronen Netzwerk keine Instanz gibt, die jedem Knoten netzwerkweit monoton wachsende Sequenznummern bereitstellt. Der Primärschlüssel für den Vergleich muss daher aus mehreren Teilschlüsseln zusammengesetzt werden. Am besten geeignet ist hier die Kombination aus Sequenznummer und Knoten ID des Senders. Damit kann nun jedes Paket eindeutig identifiziert werden. Damit die Information eines Pakets gespeichert wird, müssen folgende Prüfungen durchlaufen werden. Existiert ein Datenbankeintrag mit derselben Knoten ID und

1. derselben Sequenznummer, so ist dieses LSP ein Duplikat und wird ignoriert.
2. einer Sequenznummer, die um einen bestimmten Wert (in diesem Fall hat sich ein Wert von 50 bewährt) kleiner ist, als der entsprechende Wert im *Link State* Paket, so wird dieses Paket ebenfalls verworfen (vgl. Abbildung 11). Es muss vor kurzem ein *Master Reset* (siehe *Empfang von Master Reset Paketen*) stattgefunden haben, bei dem dieses LSP offenbar eine längere Route zum Ziel hatte und ist daher älter als das *Master Reset* Paket.
3. einer höheren Sequenznummer, so ist dieses Paket ebenfalls alt und wird abgewiesen.

Nur wenn die Sequenznummer in der Datenbank niedriger ist als im Paket (und zwar um einen kleineren Wert als in Punkt 2), wird das Paket akzeptiert. Im nächsten Schritt werden alle Einträge mit der Absender ID aus dem LSP aus der Datenbank gelöscht. Dies ist notwendig, da nicht garantiert werden kann, dass die Zahl der Einträge immer zunimmt. Im Falle eines Ausfalls von Knoten reduziert sich diese Zahl. Die neuen Einträge werden nun an die freien Stellen in der Datenbank gespeichert.

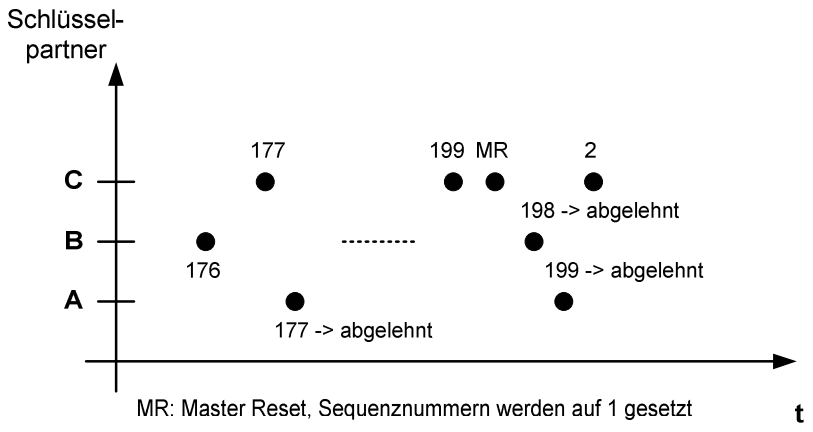


Abbildung 11 - Auswirkung eines Master Resets

- Empfang von Master Reset Paketen

  Der Empfang dieser Pakete bedeutet, dass beim Sender des Pakets die Sequenznummer übergelaufen ist und wieder von vorne beginnt. Damit jetzt nicht alle nachfolgenden Pakete aufgrund einer niedrigeren Sequenznummer verworfen werden, muss die Datenbank zurückgesetzt werden. Dazu wird die Sequenznummer aller Einträge mit der ID des Absenders wieder zurück auf 1 gesetzt. Von nun an werden nur noch *Link State* Pakete akzeptiert, deren Sequenznummer nur um einen bestimmten Wert größer ist als der aktuelle (vgl. Punkt 2 *Empfang von LSPs*). Der Verlust eines Master Reset Pakets wird derzeit nicht abgefangen. In diesem Fall werden alle weiteren Link State Pakete verworfen, bis das nächste Master Reset Paket gesendet wird.

Durch diese Schritte erhält jeder Knoten die vollständige Sicht auf das Netzwerk. Erst danach wird der Maxflow Algorithmus zur Pfadsuche beziehungsweise dem Schlüsselaufbau verwendet. Es wäre denkbar zur Pfadsuche auch andere Algorithmen zu verwenden. Diese bauen auf der vorhandenen Sicht auf das Netzwerk auf und führen die Berechnungen zur Pfadsuche dann ebenfalls lokal durch.

6.3.2 Pfadsuche

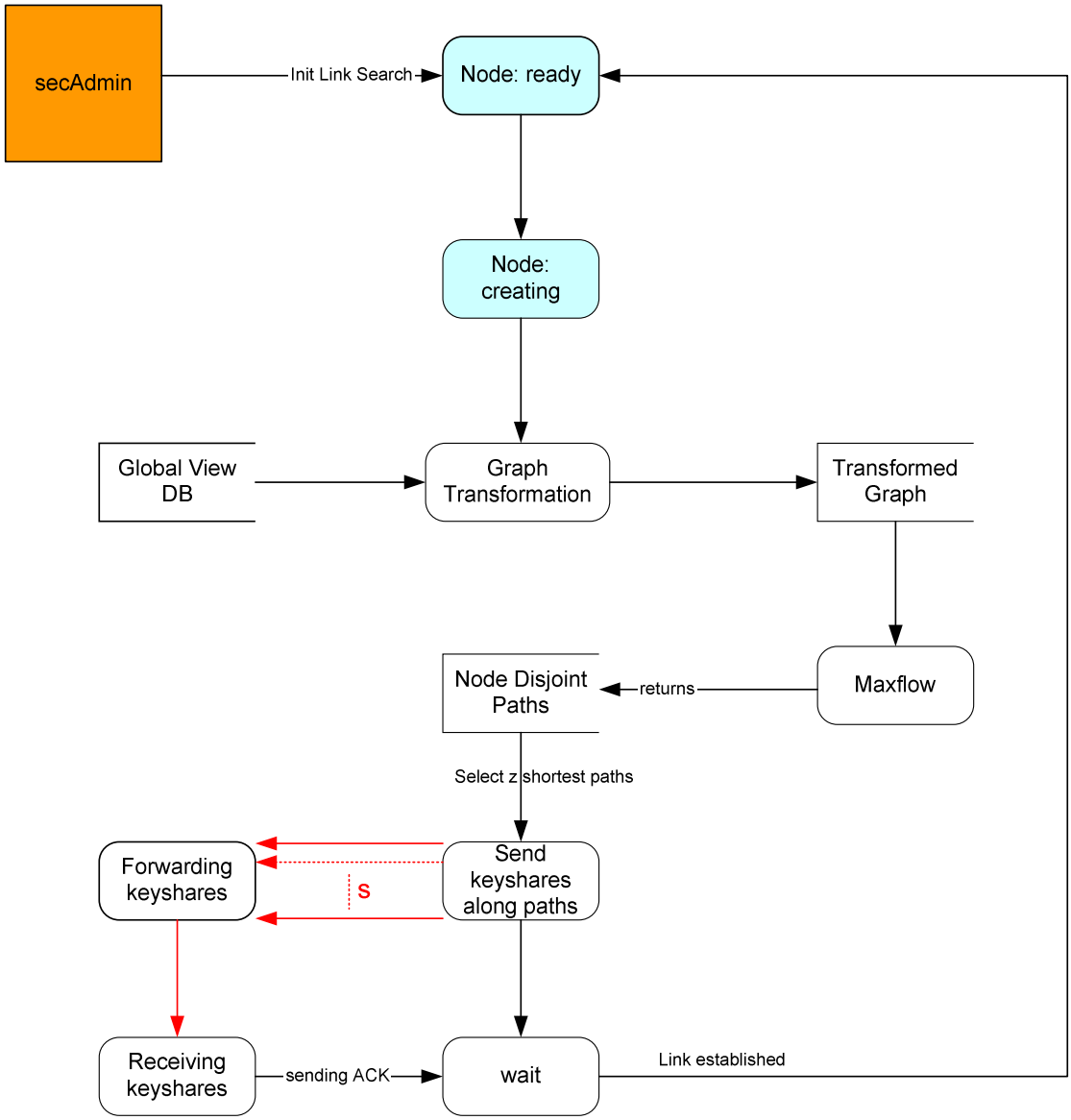


Abbildung 12 - Pfadsuche in Maxflow (LSP)

Wie in der vorherigen Phase, wird auch hier die Aktion über das Administrationswerkzeug initiiert. Dies simuliert die Notwendigkeit zur Kommunikation zweier Sensoren, die über keinen gemeinsamen Schlüssel verfügen und daher nicht direkt miteinander kommunizieren können. Dies wird durch folgende Schritte realisiert:

- Vorbereitungen (*Node: creating*)

  Nachdem das Administrationswerkzeug die Knoten ID – zu dem der Schlüssel aufgebaut werden soll – an den Knoten weitergegeben hat, wechselt er in den Zustand „Erstellen“ (*creating*). In diesem Zustand steht er anderen Knoten für das Imprinten nicht zur Verfügung. *Link State* Pakete werden in dieser Zeit jedoch noch angenommen und neue Informationen daraus in der lokalen Datenbank gespeichert.

- Graph Transformation

  Ohne eine geringfügige Anpassung an das G-SPAN Szenario lässt sich der Maxflow Algorithmus nicht verwenden. Der Algorithmus benötigt einen gerichteten Graphen, den es in einem sicheren Funknetz nicht gibt, da hier jede Verbindung zur Bestätigung der Pakete bidirektional sein muss. Das heißt, dass jede Verbindung nun durch zwei gerichtete Kanten dargestellt wird.

  Für die Berechnung disjunkter Pfade wird die Kapazität c aller Kanten $e \in E$ auf $c(e) = 1$ gesetzt. Dadurch kann jede Verbindung im Netzwerk nur einmal verwendet werden und der Algorithmus liefert die maximale Anzahl disjunkter Pfade. Leider sind diese Pfade nur kanten- und nicht knotendisjunkt. Dies bedeutet aber, dass mehrere Teilschlüssel denselben Knoten passieren können und ein Angreifer damit den vollständigen Schlüssel konstruieren kann. Um den Algorithmus dennoch einzusetzen, muss der Graph weiter umgebaut werden. Dazu werden die Knoten in je zwei virtuelle Knoten gesplittet. Ein Knoten erhält alle eingehenden Verbindungen sowie eine gerichtete Kante zu seinem Schwesterknoten. Dieser Knoten erhält zudem auch alle abgehenden Verbindungen. Die Belegung der Kapazität der Kante zwischen Schwesterknoten mit dem Wert $c(e) = 1$ garantiert, dass auch diese Kante nur einmal verwendet werden kann. Somit kann auch der „vollständige“ Knoten nur für einen Pfad herangezogen werden (vgl. [48]). Dies wird durch die nachfolgende Abbildung verdeutlicht.

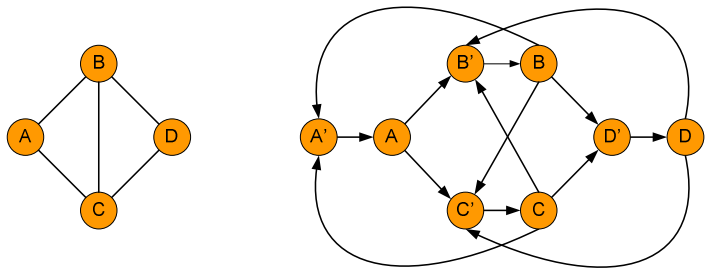


Abbildung 13 - Graph Transformation (links: ungerichtet, rechts: gerichtet)

Die beschriebenen Transformationsschritte haben unmittelbaren Einfluss auf die Effizienz der Algorithmen, da sie den Speicherbedarf in den Knoten deutlich erhöhen. Die Anzahl der Knoten verdoppelt sich durch die Teilung der Knoten in Schwesterknoten. Dadurch, dass der Algorithmus nur auf gerichteten Kanten läuft, verdoppelt sich ebenso die Anzahl der Kanten. Hinzu kommen noch die Kanten, die neu eingefügt werden müssen um die Schwesterknoten zu verbinden. Dadurch ergibt sich folgender Mehraufwand:

$$n(trans.) = 2 * n$$
$$m(trans.) = n + 2 * m$$

Um die beschriebenen Transformationsschritte durchzuführen, wird eine temporäre Datenbank angelegt, in der der umgeformte Graph gespeichert wird. Möglichkeiten zur Einsparung an Speicherplatz bieten sich durch folgende weitere Modifikation der Umformung. Durch das Löschen bestimmter Kanten wird der Wert des Maximalflusses und somit auch die Anzahl knotendisjunkter Pfade nicht verändert. Eingehende Kanten am virtuellen Startknoten sowie ausgehende Kanten des Zielknotens können nicht auf einem gesuchten Pfad liegen, da sie nicht „rückwärts“ traversiert werden können. Damit diese Knoten auf einem „Vorwärtspfad“ liegen, muss vorher entweder der Start- oder Zielknoten durchlaufen werden. Somit spielen diese Kanten und damit auch die beiden Kanten zwischen den Schwesterknoten bei Quelle und Senke bei der Pfadsuche keine Rolle Die nachfolgende Abbildung und Tabelle veranschaulichen, welche Kanten entfallen können, wenn der Graph für eine Pfadsuche von A nach D umgebaut wird.

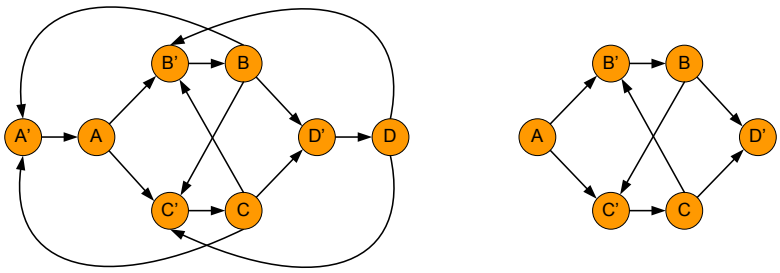


Abbildung 14 - Modifikation zur Einsparung von Kanten

Anzahl	Quelle	Ziel
1	Virtueller Startknoten	Startknoten
$\geq z$	Beliebig (eingehende Kanten)	Virtueller Startknoten
1	Virtueller Zielknoten	Zielknoten
$\geq z$	Zielknoten	Beliebig (ausgehende Kanten)

- Pfadsuche (*Maxflow*)

  Der *Maxflow* Algorithmus arbeitet nach Abschluss der Transformation nun auf der neuen Graphenstruktur. In einem $n \otimes n$ großen Feld wird für jede gerichtete Kante zwischen zwei Knoten $1 = c(e)$ in das entsprechende Feld geschrieben. Danach wird unter Verwendung der Breitensuche der maximale Fluss vom Startknoten zum Zielknoten gesucht. Da jede Kante durch eine „1“ symbolisiert wird, findet der Algorithmus mindestens z knotendisjunkte Pfade.

- Schlüsselaufbau (*send keyshares*)

  Wurden im vorherigen Schritt durch den Algorithmus mehr als z Pfade gefunden, so werden davon die **kürzesten** ausgewählt und die übrigen verworfen. Daraufhin werden die Pakete ähnlich wie bei DSR [49] verschickt. Dies hat zur Folge, dass z Pakete, welche jeweils einen Teilschlüssel sowie eine Knotenliste (*nodelist*) mit dem Pfad zum Zielknoten enthalten, erstellt und an den jeweils ersten Knoten im Pfad geschickt werden.

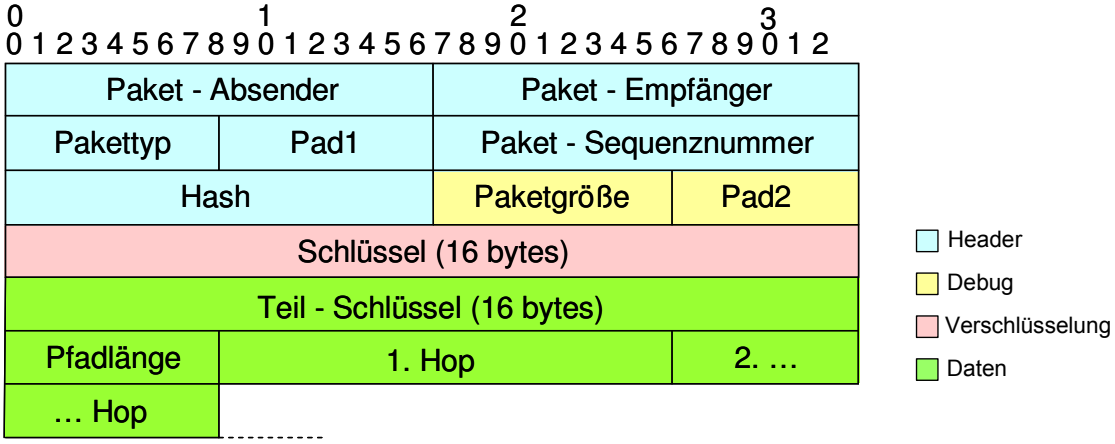


Abbildung 15 - Aufbau von Schlüsselpaketen

Die Teilschlüssel, die in den Paketen verschickt werden, werden im Schlüsselspeicher abgelegt und nacheinander miteinander „verXORt“. Die maximale Pfadlänge ist durch die Anzahl der speicherbaren Knoten im Paket bestimmt:

$$max.\ Pfadlänge = \frac{Paketgröße - Headergröße - Schlüssel - Pfadlänge}{2}\ (in\ bytes)$$

Danach geht der Knoten in eine Wartephase bis eine Empfangsbestätigung vom Zielknoten eingeht. Ein Vorteil dieser Vorgehensweise ist, dass die Teilschlüssel von den Knoten nur weitergeleitet und nicht gespeichert werden müssen. Dadurch kann ein höherer Grad an Parallelität erzielt werden, da der Schlüsselspeicher bei nicht an der Pfadsuche beteiligten Knoten nicht belastet wird.

Für die Abschätzung der maximalen Pakete für die Pfadsuche sind zwei Werte entscheidend. Dies ist zum einen die maximale Entfernung zweier Knoten (vgl. Kapitel 6) und zum anderen die Anzahl knotendisjunkter Pfade. Für den ungünstigsten Fall gilt nun, dass z Teilschlüssel immer über Pfade mit der maximalen Entfernung verschickt werden müssen. Somit ergibt sich eine maximale Anzahl von

$$\begin{aligned} max\ Pakete &= z * maximale\ Entfernung \\ &= z * \frac{n}{z} = n \end{aligned}$$

Paketen bei der Pfadsuche.

- Schlüsselumgang (*keyshare handling*)

  Erhält ein Knoten ein Paket, wird überprüft, ob das Paket tatsächlich für ihn bestimmt war, indem er seine Knoten ID in der Knotenliste sucht. Ist dies nicht der Fall, wird das Paket verworfen. Andererseits wird das Paket an den nächsten Knoten im Pfad weitergeleitet. Erreicht ein Schlüsselpaket den Zielknoten, so wird der Schlüssel gespeichert und nach seiner Vollständigkeit der Erhalt mit einer Bestätigungsnachricht beim Startknoten quittiert. Nach Empfang dieser Nachricht wechselt der Startknoten von Zustand „Erstellen“ wieder zurück auf „Bereit“. Der Schlüssel kann nun dazu verwendet werden um mit dem Zielknoten sicher zu kommunizieren.

6.3.3 Nachrichtenaufwand

Es ist nicht möglich, für die *Link State* Pakete eine konkrete Aussage über eine Obergrenze zu machen, da dieser Pakettyp periodisch verschickt wird. Im Mittel speichert jeder Knoten $2*z$ Schlüssel. Wird der Aufbau des Netzwerks vernachlässigt, so ist es jedoch möglich, eine Obergrenze für die verschickten Pakete pro Periode zu nennen:

$$max\ LSP / Periode = 2*z*n$$

Ähnlich verhält es sich mit den *Master Reset* Paketen. Sie sind abhängig von der Wiederholungsrate der LSPs. Mit jedem Überlauf der Sequenznummer muss ein neues *Master Reset* Paket verschickt werden:

$$max\ Master\ Reset : p*(2*z*n) > max\ Sequenznummer$$

Daher können zusammenfassend nur Aussagen über die Anzahl der Pakete bei der Pfadsuche gemacht werden. Durch die Verwendung von UDP ist es auf einfache Weise möglich, Pakete mit variabler Länge zu verschicken. Da in Maxflow Algorithmen *Link State* Pakete dominieren, fallen die anderen Pakettypen nicht ins Gewicht. Durch die unregelmäßige Struktur, die durch das zufällige Imprinten entsteht, lässt sich keine genaue Aussage über die Anzahl der Schlüsselpartner in den LSPs treffen. Daher wird für die Abschätzung von einer maximalen Paketgröße ausgegangen.

Pakettyp	**Paketgröße**	**Anzahl**
Link State Pakete	106 Bytes	*nicht möglich*
Master Reset Pakete	28 Bytes	*nicht möglich*
Schlüsselpakete	85 Bytes	pn
Schlüsselbestätigung	28 Bytes	p

6.3.4 Speicherverbrauch

Die Protokolle müssen auf Ressourcen-schwachen Geräte ihren Dienst tun. Die nachfolgende Tabelle gibt eine Übersicht über den benötigten Speicher.

Programm – Modul	**Speicher (Aufbau)**	**Speicher (Pfadsuche)**
Lokale Datenbank (Index, Start, Ziel)	$3m*5\ Bytes$	
Transformation virtueller / realer Knoten	$3m*2\ Bytes$	
Transformierter Graph (Start, Ziel)		$(n+2m)*4\ Bytes$
Pfadsortierung		$n*2\ Bytes$
Matrix für Kapazitäten		$n^2*2\ Bytes$
Matrix für Flüsse		$n^2*2\ Bytes$
Markierung der Pfade		$n*2\ Bytes$
Speicherung der Vorgängers für BS		$n*2\ Bytes$
Warteschlange für Pfade		$n*2\ Bytes$
Σ	$21m\ Bytes$	$4n^2+12n+8m\ Bytes$

Durch Ausdrücken von m durch n (siehe Kapitel 6) lässt sich der Gesamtspeicherbedarf für bestimmte z und n Werte bestimmen:

$$\begin{aligned} Speicherbedarf &= 21m+4n^2+12n+8m \\ &= 29\left((n-z)*z+\frac{z(z-1)}{2}\right)+4n^2+12n \end{aligned}$$

Beispiel: 100 Knoten (z=2, 197 Kanten)

Speicher für lokale Datenbank : $21*197=4137\ Bytes$

RAM für Graphtransformation : $4*100^2+12*100+8*197=42776\ Bytes$
$=\sim 41{,}8\ KB$

6.4 Maxflow (GV)

Diese Variante unterscheidet sich von der LSP Variante nur im Hinblick auf den Aufbau der globalen Netzwerksicht in der Setupphase. Daher wird in diesem Kapitel auch nur auf die Unterschiede zur LSP Variante eingegangen.

Während das Wissen über die Netzwerktopologie bei der vorher vorgestellten Variante durch periodisches Senden der Nachbarlisten erstellt wird, beziehen die Knoten sie in dieser Variante durch eine Kombination von *Link State* und *Global View* (GV) Paketen. Die *Link State* Pakete sind identisch zu der vorher besprochenen Version.

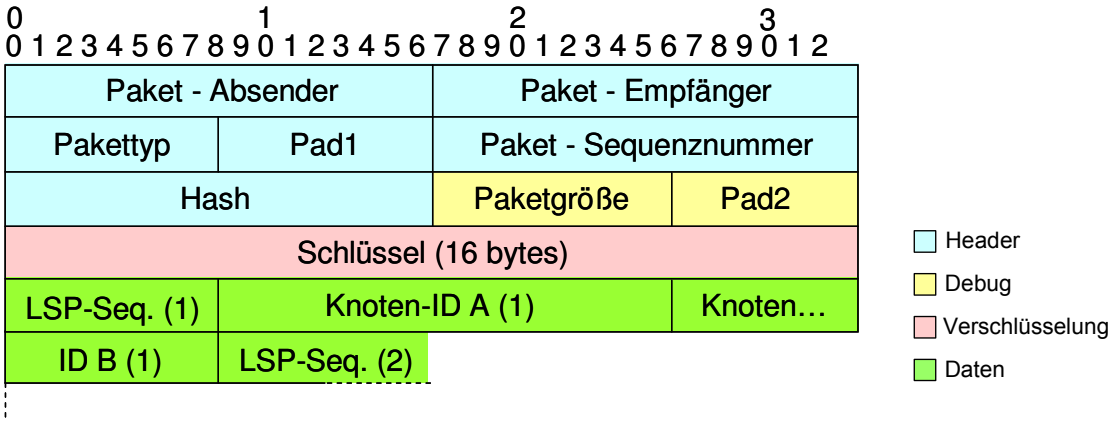


Abbildung 16 - Global View Paketaufbau

Die GV Pakete enthalten nicht nur die Schlüsselpartner des jeweiligen Knoten, sondern enthalten alle Verbindungen, die in der Datenbank des Partnerknotens gespeichert sind. Ein Eintrag in einem GV Paket besteht aus drei Komponenten:

1. der Sequenznummer des zuletzt gespeicherten *Link State* Pakets von
2. Knoten A (gespeichert wird hier die ID von Knoten A) und dessen Schlüsselpartner
3. Knoten B (gespeichert wird ebenfalls die ID)

Die damit übertragenen Daten entsprechen damit der Struktur eines Eintrags in der lokalen Datenbank. Jeder Eintrag belegt 5 Bytes Speicher, woraus sich die maximale Anzahl Einträge bei einer gewählten Paketgröße bestimmen lässt:

$$Anzahl\ Einträge = \frac{Paketgröße - Headergröße}{5} - 2\ (in\ Bytes)$$

Übersteigt die Anzahl der Einträge in der lokalen Datenbank die maximale Anzahl Einträge im GV Paket, so werden diese fragmentiert und dann hintereinander an den Zielknoten geschickt. Zum Abschluss wird noch ein GV Done Paket (siehe

Abbildung 17) versendet. Dieses signalisiert dem Zielknoten, dass seine Sicht komplett ist und keine weiteren Pakete folgen. Das GV Done Paket enthält noch die Anzahl der in der Datenbank des Senders gespeicherten Kanten. Dadurch kann überprüft werden, ob alle GV Pakete angekommen sind und somit die Sicht auf das Netzwerk mit der des Senders übereinstimmt oder ob Pakete verloren gegangen sind.

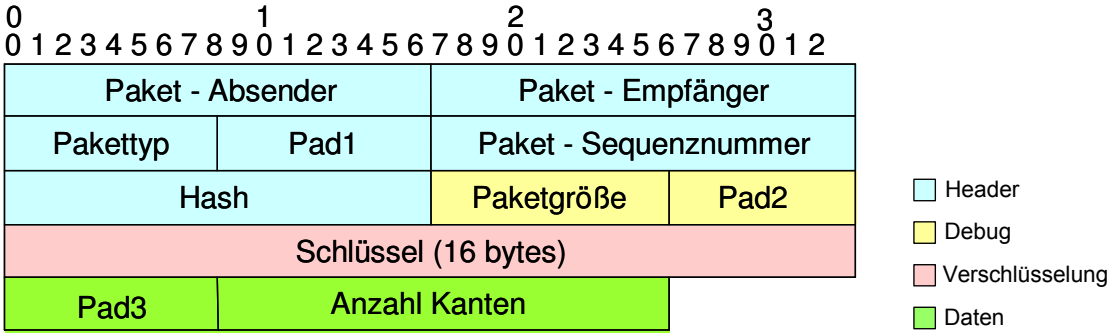


Abbildung 17 – GV Done Paketaufbau

Sind die Verbindungen aller Knoten im Netzwerk in der lokalen Datenbank eines Knotens gespeichert, werden an diesen Knoten keine weiteren GV Pakete verschickt. Dies schließt jedoch nicht aus, dass er von anderen Knoten GV Anfragen erhält.

Durch den Verzicht auf periodisches Senden von Paketen kommt diese Variante mit einer deutlich geringeren Anzahl an LSP Paketen aus. Abbildung 18 erläutert den genauen Ablauf und die einzelnen Phasen dieser Version:

6.4.1 Setup

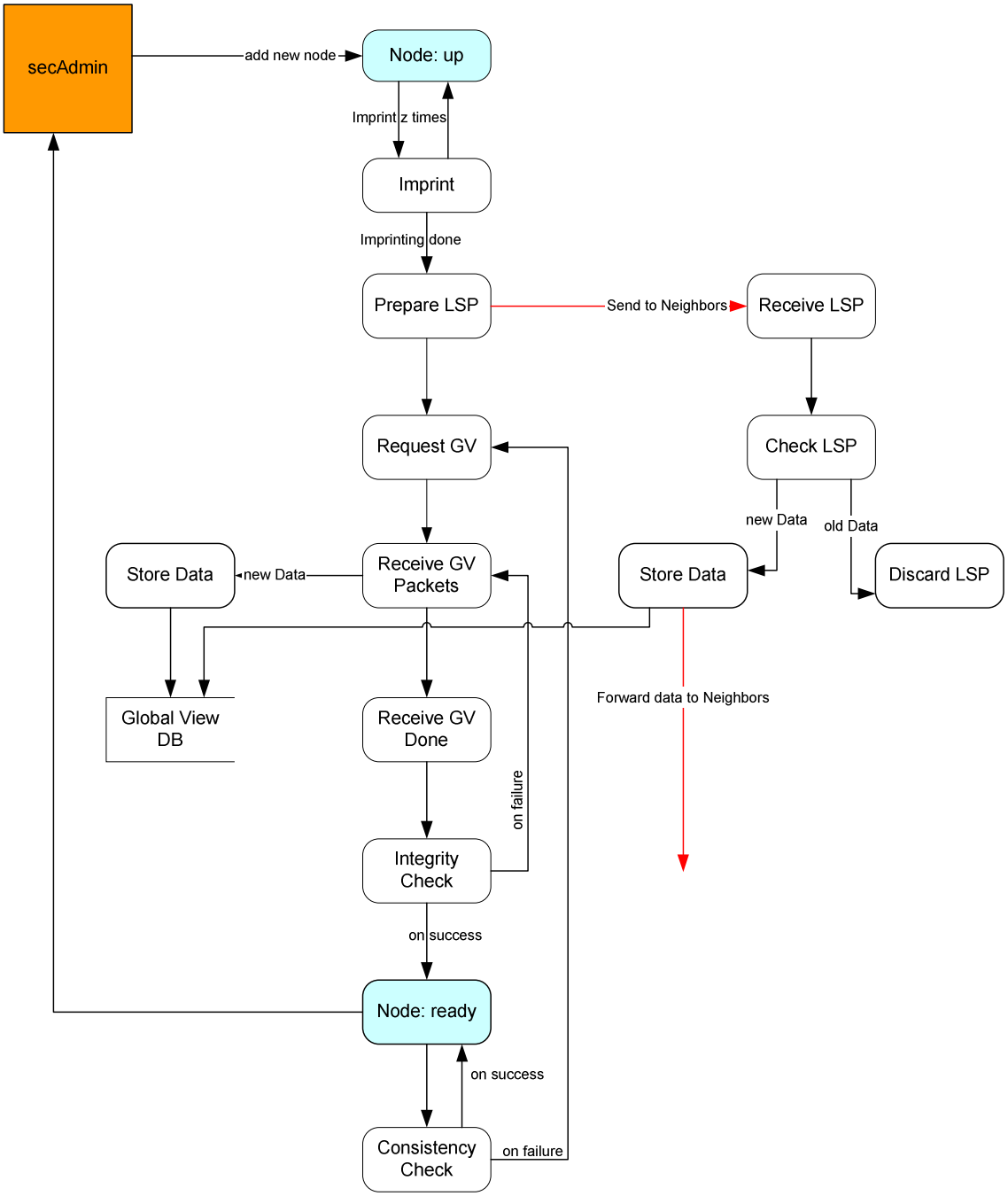


Abbildung 18 – Flussdiagramm Maxflow (GV)

Mit Hilfe des Administrationswerkzeugs wird ein neuer Knoten erstellt. Dessen Initialzustand lautet *up*. Das bedeutet, dass er in diesem Zustand anderen Knoten noch nicht bekannt ist und ihnen somit für Funktionen wie Schlüsseltransport nicht zur Verfügung steht. Der Knoten muss selbst aktiv werden und in den Zustand *ready* wechseln. Dazu müssen folgenden Schritte ausgeführt werden:

- Initialer Schlüsselaufbau (*Imprinten*)

  Wie bei allen Verfahren muss zu Beginn der Schlüsselspeicher des Knoten mit mindestens z Schlüsseln aufgefüllt werden um mit anderen Knoten in Kontakt treten zu können.

- Bekanntmachen im Netzwerk (*LSP Handling*)

  Danach muss der Knoten alle anderen Knoten im Netzwerk über seine Verfügbarkeit informieren. Dies geschieht auch hier durch das Versenden von *Link State* Paketen, die eine Liste aller gültigen Schlüssel enthält. Im Gegensatz zur vorherigen Variante werden diese Informationen einmalig und nur in diesem Zustand an die Nachbarknoten verschickt. Die maximale Anzahl von *Link State* Paketen, die zum Aufbau eines Netzwerks notwendig sind lässt sich durch folgende Gleichung bestimmen:

$$\begin{aligned} max\ LSP &= \sum_{i=z+1}^{n} z\,((i-1)\,z-1) \\ &= \sum_{i=z+1}^{n} z^2\,(i-1) - \sum_{i=z+1}^{n} z \end{aligned}$$

  Da die Knoten erst ab z Schlüsseln im Schlüsselspeicher anfangen LSPs zu verschicken, beginnt der Index erst mit dem um 1 erhöhten Wert. Mit dazugekommenen Knoten wird das Netz durch diesen mit LSPs geflutet. Dabei kann jedes LSP genau z mal über eine Verbindung geschickt werden. Einmal beim Senden und $z-1$ mal beim Empfangen. Diese Anzahl multipliziert mit der jeweiligen Anzahl Kanten ergibt somit die maximal mögliche Anzahl geschickter *Link State* Pakete.

- Anfordern der Netzwerkstruktur (*GV Request*)

  Nach dem Versand der LSPs wird vom jüngsten Schlüsselpartner die Netzwerksicht angefordert. Dies ist notwendig, um die lokale Datenbank mit allen im Netzwerk vorhandenen Einträgen zu vervollständigen.

  Die einzige Bedingung für den Versand von GV Request Paketen stellt die Speicherung von mindestens z Schlüsseln dar. Dies ist bei den ersten z Knoten nicht gegeben, weshalb für sie eine Sonderbehandlung eintritt und diese Knoten kein GV Request Paket verschicken:

$$max\ GV\ \ Requests = n - z$$

- Verarbeiten der Netzwerkinformationen (*GV Handling*)

  Beim Erhalt der GV Pakete wird jeder Eintrag mit der lokalen Datenbank verglichen. Doppelte Einträge werden verworfen, neue Einträge werden gespeichert.

- Überprüfung der Daten (*Integrity Check*)

  In einem Netzwerk können aus verschiedenen Gründen Nachrichten verloren gehen. Da in diesem Verfahren jedoch keine Nachrichten redundant sind, findet nach dem Erhalt der GV Done Nachricht eine Vollständigkeitsprüfung statt. Im GV Done Paket ist die Anzahl der gespeicherten Kanten beim Absender gespeichert. Stimmt diese Zahl nicht mit der Anzahl in der lokalen Datenbank gespeicherten Verbindungen überein, wird erneut ein GV Request an den Schlüsselpartner gesendet.

- Konsistenzprüfung der Daten (*Consistency Check*)

  Treten während der Übertragung von GV Paketen Paketverluste auf (zum Beispiel aufgrund von Kollisionen auf dem Medium oder durch Überlauf der Warteschlange eines Knotens), wird das Paket verworfen. Dieser Fall wird durch die Integritätsprüfung abgefangen. Gehen aber *Link State* Pakete verloren, so fehlen wichtige Informationen bezüglich der Netzwerksicht. Ein Knoten, der sich an diesem imprintet, würde daher auch nur eine eingeschränkte Sicht auf das Netzwerk bekommen. Der Integritätscheck schlägt keinen Alarm, da auch der Knoten, der das GV Request Paket bekommt, keine vollständige Sicht hat.

Um dieses Problem zu lösen, werden die PING Nachrichten „missbraucht“. Sie werden in unregelmäßigen Abständen zwischen den Schlüsselpartnern hin- und hergeschickt. Auf einen PING antwortet der Knoten mit einer PONG Nachricht. Primär dienen diese Nachrichten dazu, die Betriebsbereitschaft beziehungsweise das „Vorhandensein“ von Geräten zu überprüfen. Des Weiteren übermitteln die Knoten mit einem PING auch die Anzahl der Geräte und ihrer Schlüsselpartner aus der lokal gespeicherten Datenbank. Dadurch hat ein Knoten die Möglichkeit, seine Sicht mit der mehrerer Knoten abzugleichen. Erhält ein Knoten einen PING von einem Knoten mit einer höheren Anzahl von Datensätzen, sendet er ein GV Request Paket an den Knoten mit der umfangreicheren Netzwerksicht. So lange mindestens ein Knoten alle Pakete empfangen hat und somit eine vollständige Sicht auf das Netzwerk hat, so wird sich dieses Wissen mit der Zeit verbreiten bis alle Knoten dieselbe Sicht aufweisen.

Empfangsroutine

- Neuer Knoten im Netzwerk (*Link State* Pakete)

  Beim Erhalt eines *Link State* Pakets verhalten sich die Knoten ebenso wie bei der Maxflow Variante mit periodischen LSPs (siehe vorheriger Abschnitt).

- Anforderung der Netzwerkstruktur (*GV Request*)

  Nach dem Erhalt dieses GV Requests übermittelt der Schlüsselpartner an dessen Absender die globale Sicht. Diese setzt sich aus allen Einträgen, die der Knoten in seiner Datenbank hält, zusammen. Jeder Eintrag besteht dabei aus einem Index, einem Quell- und einem Zielknoten, der das Schlüsselpaar und die Periode identifiziert. Überschreitet die Anzahl der lokal gespeicherten Einträge die maximale Anzahl (bei der derzeitigen Paketgröße sind das 12), so werden die Informationen fragmentiert und in mehrere Pakete aufgeteilt. Nach dem letzten Daten-Paket wird noch ein Abschluss-Paket (GV Done) gesendet um zu signalisieren, dass keine weiteren Pakete folgen und um die Vollständigkeit der Daten prüfen zu können.

  Die Anzahl der maximal verschickten GV Pakete ist Abhängig von der Anzahl der im Netz aufgebauten Verbindungen (Kanten) und von der Zahl der maximal speicherbaren Verbindungen in einem GV Paket. Für den allgemeinen Fall gilt somit folgende Obergrenze:

$$
\begin{aligned}
max\ GV &= \sum_{i=z+1}^{n} 1 + \left\lfloor \frac{Kantenzahl - 1}{max\ GV\ \ Einträge} \right\rfloor \\
&= \sum_{i=z+1}^{n} 1 + \left\lfloor \frac{z(i-z) + z(z-1) - 1}{max\ GV\ \ Einträge} \right\rfloor \text{ [1]} \\
&= \sum_{i=z+1}^{n} 1 + \left\lfloor \frac{i(z-1) - 1}{max\ GV\ \ Einträge} \right\rfloor \\
&\leq \frac{n^2 z + n(24 - z) - z^3 + z^2 - 24z}{26}
\end{aligned}
$$

Robustheit:

Durch den *Integrity* und *Consistency Check* kann das Maxflow (GV) Protokoll Paketverluste des Typs *LSP,GV* sowie *GV Dones* tolerieren. Die übrigen Pakettypen werden zu „schwachen" Datenverkehrszeiten gesendet, so dass hier in der Praxis keine Verluste bemerkbar waren. Die Robustheit des Protokolls nimmt durch die Korrekturmaßnahmen dadurch erheblich zu. Werden die Basisfunktionen auch gegen Verlust abgesichert, lässt sich das Protokoll theoretisch in beliebig großen Netzen (unter Berücksichtigung des Speicherbedarfs) einsetzen, ohne dass Paketverluste die Funktionalität beeinflußen.

6.4.2 Pfadsuche

Die Pfadsuche ist von Link State oder GV Paketen unabhängig und ist daher äquivalent zu der LSP Variante.

6.4.3 Nachrichtenaufwand

Aufgrund der Tatsache, dass nach dem Aufbau des Netzwerks zur Erhaltung der globalen Sicht keine weiteren Pakete mehr verschickt werden müssen, lässt sich in dieser Variante eine Obergrenze für die Anzahl maximal verschickter Nachrichten angeben. Diese Werte gelten unter der Annahme, dass kein Paketverlust aufgetreten ist und Pakete nicht wiederholt gesendet werden.

Die Obergrenze für die *Link State* Pakete wurde schon in Kapitel 6.4.1 (siehe Seite 54) berechnet. Für den theoretischen Vergleich mit anderen Protokollen ist eine Abschätzung nach n notwendig, die sich wie folgt darstellt:

[1] $\lfloor x \rfloor, x \in \mathbb{R}$ liefert den ganzzahligen Wert einer Zahl

$$
\begin{aligned}
max\ LSP &= \sum_{i=z+1}^{n} z^2\ (i-1) - \sum_{i=z+1}^{n} z \\
&\leq z^2 \sum_{i=z+1}^{n} (i-1) \\
&\leq z^2 \frac{n(n+1)}{2}
\end{aligned}
$$

In dieser Variante werden keine periodischen LSPs ausgesandt. Ein Master Reset wäre in diesem Fall erst notwendig, wenn ein Knoten so oft imprintet hat, wie die Grenze zum Überlauf – in G-SPAN ~ 200 Mal – (vgl. Kapitel 6.3.3) groß ist. Da der Schlüsselspeicher davor aber aufgrund des begrenzten Speichers schon mehrfach übergelaufen wäre, tritt dieser Fall erst nach einer sehr großen Anzahl von Pfadsuchen (mit späterem Verwerfen der Schlüssel) auf.

Pakettyp	**Paketgröße**	**Anzahl**
Link State	106 Bytes	$\leq 0{,}5\ (z^2(n^2+n))$
GV Request	28 Bytes	$n-z$
GV	≤ 93 Bytes	$\leq \frac{1}{26}(n^2 z + n(24-z) - z^3 + z^2 - 24z)$
GV Done	30 Bytes	$n-z$
Schlüsseltransport	85 Bytes	pn
Schlüsselbestätigung	28 Bytes	p
Master Reset	28 Bytes	*nicht möglich*
$\sum$		$\leq n^2 z^2 + 2{,}5n \xrightarrow{z=2} 4n^2 + 2{,}5n$

6.4.4 Speicherverbrauch

Der Speicherverbrauch unterscheidet sich nicht von der Maxflow Variante mit *Link State* Paketen. Die *Global View* Pakete dienen nur zum Aufbau der globalen Sicht, haben aber keinen Einfluss auf die Graphtransformation oder den benötigten Speicher in der lokalen Datenbank der Knoten.

6.5 Tiefensuche

Um die Tiefensuche für die Pfadsuche verwendbar zu machen, müssen die gefundenen Pfade zunächst gespeichert werden. Danach wird begonnen alle Pfade auf Knotendisjunkheit zu untersuchen bis s knotendisjunkte Pfade gefunden wurden. Die in G-SPAN verwendete Maxflow Implementierung verwendet die der Tiefensuche sehr ähnliche Breitensuche. In Folge der mathematischen Grundlage (z.B. der Betrachtung des Residualnetzwerks bzw. der flussvergrößernden Pfade) in Maxflow Algorithmen unterscheiden sich die Protokolle und damit ihre Performanz deutlich voneinander. Aus einer früheren Arbeit [12] entstanden Implementierungen dreier unterschiedlicher Varianten der Tiefensuche für den G-SPAN Simulator.

Traditionelle Tiefensuche

Unter Verwendung des verteilten Tiefensuchalgorithmus schickt der initiierende Knoten eine Anfrage an all seine Schlüsselpartner. Dieses Paket enthält eine Resttiefe, die von jedem Knoten vor dem Weiterleiten des Pakets dekrementiert wird. Erreicht die Resttiefe den Wert „0“, so wird das Paket nicht mehr weitergeleitet. Falls für die Resttiefe nicht die maximale Entfernung im Netzwerk verwendet wird, ist eine erfolgreiche Pfadsuche nicht garantiert. Wird sie dagegen verwendet, so ist der Nachrichtenaufwand so immens ($m^{max\ Entfernung}$), dass selbst bei einer sehr geringen Sendedauer für ein Paket die benötigte Gesamtdauer schon in kleinen Netzwerken mehrere Minuten beträgt und somit nicht akzeptabel ist. Ein weiterer Problempunkt ist der Speicherbedarf im Knoten. Um Paketverluste auszugleichen muss in jedem Knoten der Sender der Anfrage sowie der aktuell befragte Knoten gespeichert werden. Bei einem Fluten des Netzwerks mit einer Pfadsuche benötigt jedes Gerät linearen Speicher. Dies ist bei den Ressourcen – beschränkten Sensoren nicht möglich.

Iterative Tiefensuche (*Expanding Ringsearch*)

Anstatt wie in der vorherigen Variante die Anfrage mit einer maximalen Resttiefe zu verschicken, wird in diesem Verfahren die Resttiefe (beginnend bei 1) nach jeder erfolglosen Pfadsuche inkrementiert. Dadurch wird die Anfrage bei dem ersten Fund von s knotendisjunkten Pfaden ab. Dadurch wird eine erhebliche Anzahl von Nachrichten eingespart und die Dauer für die Pfadsuche minimiert. Das Problem des hohen Speicherbedarfs für größere Tiefen bleibt jedoch bestehen.

Lokale Tiefensuche

In diesem Fall werden zur Pfadsuche keine Nachrichten an die Schlüsselpartner geschickt, sondern die Pfadsuche auf der lokalen Datenbank durchgeführt. Die Übertragung der Netzwerksicht wird in diesem Protokoll mit Hilfe von periodischen HELLO Paketen realisiert. Der Inhalt dieser Pakete ähnelt dem der *Link State* Pakete wie sie bei Maxflow verwendet werden. Ist die Pfadsuche auf der lokalen Datenbank erfolgreich, wird entlang jedes Pfads ein Teilschlüssel verschickt. Damit bietet sich ein ähnliches Bild betreffend der Messungen wie bei der LSP Variante des Maxflow Protokolls. Durch das periodische Versenden von Paketen ist keine Aussage über eine theoretische Obergrenze von Paketen zum Netzwerkaufbau möglich. Für die Pakete zur Pfadsuche sind maximal n Pakete notwendig (vgl. Schlüsselaufbau Maxflow Seite 48).

Messungen anhand der Tiefensuche können im G-SPAN Simulator nur sehr eingeschränkt betrieben werden. Die lokale Tiefensuche arbeitet mit periodischen Paketen (HELLO) weshalb es wie bei Maxflow (LSP) nicht möglich ist eindeutige Werte zu bestimmen. Die traditionelle Tiefensuche erzeugt einen immensen Datenverkehr und benötigt sehr lange bis die Pfade gefunden werden. Daher wird auf Messungen dieser beiden Varianten im Simulator verzichtet. Die iterative Tiefensuche arbeitet in kleinen Netzen noch, bei größeren Netzen kommt es zu Problemen bzw. es werden keine z knotendisjunkte Pfade gefunden. Dies liegt daran, dass der Simulator vorsieht, dass Pakete eine maximale Länge aufweisen. Dadurch ist die Länge der in einem Paket gespeicherten Pfade begrenzt. Tritt dieser Fall auf, bricht das Protokoll ab. Daher sind in G-SPAN nur Messungen bis zu einer gewissen Größe möglich.

6.6 RKEP (manuell)

Bei der Variante mit manuellem Imprinten wird davon ausgegangen, dass die Graphstruktur vom Master Device korrekt erstellt und auch erhalten wird. Daher wird in diesem Abschnitt nur auf die Nachrichten zur Kommunikation eingegangen.

Nach [43] speichert jeder Knoten lokal alle Einträge über laufende Schlüsselaufbau-Vorgänge in einem so genannten *Query Set*. Jeder dieser Einträge besteht aus 3 Komponenten:

- ID der Suchanfrage (*Query ID*)

  dient zur eindeutigen Identifikation der Anfrage

- potentielle Schlüsselpartner (*Established Set*)

  speichert alle Knoten, die einen Schlüssel zum Zielknoten aufgebaut haben

- anfragende Knoten (*Requesting Set*)

  enthält alle Knoten, die diese Query verschickt haben und keinen Schlüssel zum Zielknoten haben

Wird der Schlüsselaufbau am Startknoten getriggert, so sendet dieser Startknoten an all seine Schlüsselpartner eine Anfrage (*Query*). Empfängt ein Knoten (*Receiver*) diese Anfrage wird zuerst überprüft, ob er einen Schlüssel zu dem Zielknoten (*Target*), der in der Anfrage angeben ist, bereitstellt. Ist dies der Fall, so wird dem Knoten über den die Anfrage empfangen wurde (*Sender*), eine Establish Nachricht geschickt. Ist dem Receiver der Zielknoten ebenfalls unbekannt, so wird als nächstes überprüft, ob die Anfrage schon einmal über einen anderen Knoten empfangen wurde. Trifft dies zu, so wird der Sender mit in das Requesting Set dieser Anfrage übernommen. Damit speichert ein Knoten, an wen im Fall einer erfolgreichen Pfadsuche eine Nachricht geschickt werden soll. Hat der Receiver diese Anfrage zum ersten Mal empfangen, so startet er seinerseits einen Schlüsselaufbau zum Zielknoten. Dazu wird das *Query Set* um die neue Anfrage erweitert und an alle Schlüsselpartner außer dem Receiver weitergeleitet. Durch dieses Verhalten wird die Anfrage an jeden Knoten im Netzwerk geschickt, bis sie die 1-Hop Nachbarn des Zielknoten erreicht. Diese haben einen gültigen Schlüssel zum Zielknoten und beginnen den Sendern der Anfrage Establish Nachrichten zu schicken. In diesen Nachrichten ist die ID des Knotens, der einen Schlüssel zum Zielknoten unterhält gespeichert, ebenso wie die ID der Anfrage, die nach diesem Zielknoten gefragt hat.

Erhält ein Knoten nun eine Establish Nachricht, so wird überprüft, ob er überhaupt eine Anfrage zu diesem Zielknoten verschickt hat. Ist dies nicht der Fall, wird die Nachricht verworfen und der Knoten wechselt in den *ready* Zustand. Ist die Anfrage aber in seinem *Query Set*, fügt sich der Receiver zum Established Set dieser Anfrage

hinzu und entfernt sich gleichzeitig aus dem Requesting Set. Ist die Anzahl der Einträge in dem Established Set des Knotens größer als z, baut der Knoten einen Schlüssel zum Zielknoten auf. Nach einem erfolgreichen Schlüsselaufbau schickt der Receiver seinerseits eine Established Nachricht an seine Schlüsselpartner (ausgenommen dem Sender der letzten empfangenen Established Nachricht).

Damit der Schlüsselspeicher der Knoten nicht voll läuft, verschickt jeder Knoten nach einem erfolgreichen Schlüsselaufbau eine Nachricht zum Schlüsselabbau (*Cancel Query*) an alle Knoten, die mit dieser Anfrage schon einmal einen Schlüssel zum Zielknoten aufgebaut haben.

Durch diese Vorgehensweise baut jeder Knoten angefangen bei den 2-Hop Nachbarn des Zielknotens einen Schlüssel zu demselben auf. Nachdem die 3-Hop Nachbarn einen Schlüssel aufgebaut haben, verwerfen die 2-Hop Nachbarn ihre Schlüssel. Dieses Verfahren wird fortgesetzt bis der Startknoten erfolgreich einen Schlüssel zum Zielknoten aufgebaut hat.

6.6.1 Nachrichtenaufwand

Für RKEP lässt sich die theoretische Obergrenze für die Paketanzahl ebenfalls bestimmen. Für die Anzahl der Kanten m in z-verbundenen Graphen gilt dieselbe Berechnungsformel wie in Kapitel 6. Um die Größe der verschickten Pakete zu bestimmen, muss bekannt sein, welche Informationen mitverschickt werden. Zu dem Header von 28 Bytes/Paket kommen für die *Query ID* noch 6 Byte bei *Query*, *Established*, *Valid* und *Cancel* Paketen hinzu. Einzig in den Forward Nachrichten werden weitere Information (der Zielknoten (2 Bytes) und der Teilschlüssel (16 Bytes) benötigt. *Link Drop* Pakete bedürfen keiner weiteren Information.

Pakettyp	**Paketgröße**	**Anzahl**
Queries	34 Bytes	$\leq 2\,m$
Established	34 Bytes	$\leq m - \frac{1}{2}(s^2 + s) \leq m$
Valid	34 Bytes	$\leq n - \frac{1}{2}(s^2 + s) \leq n$
Cancel Query	34 Bytes	$\leq m$
Forward Keyshare	52 Bytes	$2\,v$
Link Drop	28 Bytes	*nicht möglich* [2]
$\sum$		$\leq n\,(4z+3) - 2z(z+1) \xrightarrow{z=2} 11n$

[2] aufgrund des randomisierten Imprintens kann nicht bestimmt werden, wann wie viele Knoten aus Speicherplatzproblemen einen Schlüssel verwerfen müssen, der Faktor ist in der Praxis jedoch minimal

Es ist leicht nachvollziehbar, dass das Paketaufkommen für einen Schlüsselaufbau stark von der Graphstruktur abhängig ist. Abhängig von der Position des Zielknotens gibt es Strukturen, die verhindern, dass für eine *Query* das ganze Netzwerk geflutet wird.

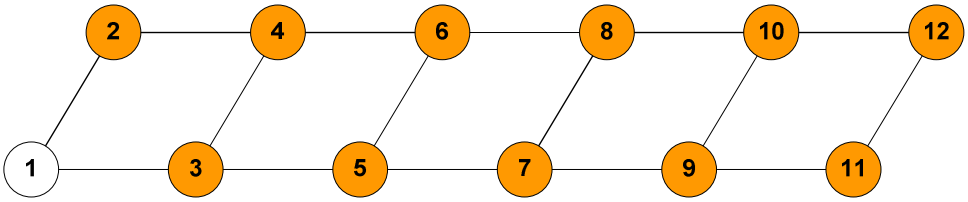


Abbildung 19 - RKEP Kette

Abbildung 19 verdeutlicht, welche Auswirkungen die Wahl des Zielknotens mit sich bringt. Sei Knoten 1 der Knoten, der die *Query* initiiert. Sei Knoten 12 der Zielknoten. Dann wird die *Query* an alle Knoten einschließlich Knoten 10 und 11 weitergeleitet bis letztere eine Establish Nachricht in Richtung Knoten 1 zurückschicken. Angenommen, bei dem Zielknoten handelt es sich um Knoten 7. Dann werden die *Query* Nachrichten nur bis Knoten 4 und 5 geschickt. Dort „bricht die Kette ab", das Netz wurde nur bis zur Hälfte geflutet. Bei zunehmender Knotenanzahl wird die Kette immer länger und die Wahrscheinlichkeit, dass sich der Zielknoten im mittleren Bereich des Graphen befindet steigt. Für RKEP ist die Kettenform des Graphen damit um eine vorteilhafte Konfiguration bezüglich des Nachrichtenaufwands.

6.6.2 Speicherverbrauch

Da es sich bei RKEP um ein verteiltes Protokoll handelt, fällt der Speicherbedarf bei den einzelnen Knoten im Vergleich zu den bisherigen Protokollen deutlich geringer aus. RKEP speichert pro Schlüsselaufbau ein *Query Set*. Dieses besteht aus

Query Set	**Speicher**
Query ID (Zähler, ID von Start- und Zielknoten)	$3*2$ Bytes
Established Set (Knoten ID)	$2*Anzahl(ES)$ Bytes
Requesting Set (Knoten ID)	$2*Anzahl(RS)$ Bytes
Σ	$6+Anzahl(Nachbarn)*2$ **Bytes**

Knoten, die im Established Set gespeichert sind, können nicht gleichzeitig im Requesting Set gespeichert sein und umgekehrt. Entweder verfügen die Knoten über einen Schlüssel oder sie fragen ihn an. Daher schließen sich die beiden Sets aus. Die Anzahl der Knoten beider Sets kann somit die Anzahl der Nachbarn nicht überschreiten. Die Anzahl der Nachbarn lässt sich nicht für unmittelbar für alle Knoten bestimmen. Für die Berechnungen wird von einem Mittelwert von $2*z$ Nachbarn ausgegangen.

Beispiel: 100 Knoten (z=2, 197 Kanten)

lokaler Verbrauch (1 Knoten) : $6+4*2=14\ Bytes$

Gesamtverbrauch (alle Knoten) : $100*14=1400\ Bytes$

6.7 RKEP (automatisch)

Diese Variante unterscheidet sich von RKEP manuell nur durch den Imprint Vorgang. Daher wird in diesem Kapitel auch nur auf die Unterschiede zu dieser Version eingegangen.

In dieser Implementierung wird davon ausgegangen, dass kein Master Device verwendet wird. Die Knoten müssen also selbständig überprüfen, ob sie sich an den „richtigen“ Knoten imprintet haben. Unter richtig wird in diesem Fall ein Imprinten verstanden, welches eine Graphstruktur gemäß Kapitel 5.3 erstellt.

Um sicherzustellen, dass sich der neue Knoten an z benachbarte Knoten imprintet hat, fordert dieser von seinen Schlüsselpartnern Nachbarlisten an. In diesen Listen, sind die Schlüsselpartner der jeweiligen Knoten enthalten. Enthält die Schnittmenge aller Nachbarlisten, genau die Knoten, an denen imprintet wurde, so handelt es sich um einen z-Connector und das Imprinting für diesen Knoten ist beendet. Sind nicht alle Knoten benachbart, so werden $z-1$ Schlüssel verworfen. Der Schlüssel des Knotens mit der kleinsten Nachbarliste (*Root*) wird beibehalten. Im 2. Durchlauf imprintet sich der neue Knoten an die initialen Schlüsselpartner des „Wurzel“ (*root*) Knotens. Dadurch wird sichergestellt, dass so ein z-Connector entsteht und der Algorithmus nicht in einen undefinierten Zustand fällt.

6.7.1 Nachrichtenaufwand

Der Nachrichtenaufwand ist bis auf die Anzahl der Nachbarlisten identisch mit der manuellen Variante. In der Nachbarliste ist die Query ID (6 Bytes), die Anzahl der Knoten (1 Byte) und die Liste mit den Nachbarn (z*2 Byte) zu denen ein initialer Schlüssel erstellt wurde.

Pakettyp	Paketgröße	Anzahl
Queries	34 Bytes	$\leq 2\,m$
Established	34 Bytes	$\leq m - \frac{1}{2}(s^2 + s) \leq m$
Valid	34 Bytes	$\leq n - \frac{1}{2}(s^2 + s) \leq n$
Cancel Query	34 Bytes	$\leq m$
Forward Keyshare	52 Bytes	$2\,v$
Link Drop	28 Bytes	*nicht möglich* [3]
Request Neighborlist	28 Bytes	$\leq n * z$
Neighborlists	35 + 2z Bytes	$\leq n * z$
$\sum$		$\leq n\,(4z+3) - 2z(z+1) + 2n * z \xrightarrow{z=2} 15n$

6.7.2 Speicheraufwand

Bezogen auf den Speicheraufwand, zeigt die Variante mit Neighborlist Paketen keinen Unterschied zu RKEP mit manuellem Imprinten. Es gelten daher dieselben Werte wie in Kapitel 6.6.2.

[3] siehe Kapitel 6.6.1

7 Analyse

Dieses Kapitel gibt Aufschluss über die Szenarien, die ausgewählt wurden um die Effizienz der untersuchten Algorithmen in der Praxis zu überprüfen und die Ergebnisse der theoretischen Analyse zu validieren. Die Ergebnisse dieser Tests werden im Anschluss in Form von Diagrammen veranschaulicht.

7.1 Testszenarien

Dieser Abschnitt beschreibt die unterschiedlichen Testverfahren, denen die verschiedenen Protokolle unterzogen wurden. Ein exakter Vergleich ist dabei nur möglich, wenn die Protokolle jeweils auf demselben Graphen ausgeführt werden. Dazu wurde ein Graph mit 25 Knoten generiert. Die chronologische Reihenfolge, welcher Knoten welche Schlüsselpartner hat, wird in einer externen Datei (Verlauf) gespeichert. Dadurch kann derselbe Graph durch jedes Protokoll aufgebaut werden. Der Ausgangsgraph wurde sukzessive um weitere 25 Knoten erweitert und neu untersucht bis zu einer Obergrenze von 250 Knoten.

In den folgenden Szenarien werden die verschiedenen Protokolle mit unterschiedlicher Knotenanzahl und verschieden Werten für z untersucht.

Szenario 1: Setup

In diesem Szenario wird der Aufwand an Nachrichten gemessen, die zur Entwurfszeit des Netzwerkes notwendig sind. Dies tritt nur bei Protokollen auf, die zu dieser Zeit schon Nachrichten versenden. Bei den hier untersuchten Algorithmen ist das nur bei der Maxflow Variante der Fall. Die Ergebnisse können aber verwendet werden, um sie mit später entwickelten Protokollen, die auf einer lokalen Sicht basieren, zu vergleichen. Die Maxflow (LSP) Variante sowie die lokale Tiefensuche werden aufgrund von periodischem Versand von Paketen nicht betrachtet.

Szenario 2: Neusynchronisation

In diesem Fall werden nur die Daten berücksichtigt, die anfallen, wenn zu einem bestehenden Netzwerk ein weiteres Gerät hinzugefügt wird. Laufen alle Knoten wieder synchron, das heißt, alle haben dieselbe Sicht auf das Netzwerk (Maxflow) beziehungsweise wurden korrekt neu imprintet (automatisches RKEP), wird die Messung beendet.

Szenario 3: Pfadsuche bzw. Schlüsselaufbau

Die Anzahl der Nachrichten, die notwendig sind um eine Verbindung zwischen zwei Geräten aufzubauen, stellt eine wichtige Kenngröße dar um die Effizienz von Protokollen zu messen. Gemessen werden hierbei alle Nachrichten, die verschickt werden bis beide Geräte den Schlüssel aufgebaut haben und eventuelle Bestätigungsnachrichten (*Acknowledgements*) verschickt wurden.

Für die Pfadsuche wird die zugrunde liegende Graphstruktur geändert. Die Messungen erfolgen nicht an einem zufällig imprinteten Graphen, sondern an der Kettenstruktur (vgl. Kapitel 6). Für RKEP wird zum einen ein Schlüsselaufbau über die ganze Kette (Knoten 1 bis n) gestartet – dies stellt den größten Aufwand (*worst case*) dar. Zum anderen wird noch ein Schlüsselaufbau über die Hälfte der Länge (Knoten 1 bis $n/2$) initiiert – dies liefert den durschnittlichen Wert. Bei den Maxflow Protokollen spielen die Pakete zum Schlüsselaufbau nur eine untergeordnete Rolle, da ihre Anzahl im Vergleich zu RKEP sehr gering bleibt. Daher werden für den Vergleich die Anzahl der Pakete von Knoten 1 bis n herangezogen. Die Pfadlänge überschreitet bei Maxflow und der Tiefensuche den maximal zur Verfügung stehenden Speicher im Paket. Daher werden für diese Fälle die theoretischen Werte verwendet. Die Tiefensuche liefert schon ab 50 Knoten keinen erfolgreichen Schlüsselaufbau und hat gleichzeitig schon ein Vielfaches an Paketen versendet. Die Tiefensuche wird daher in den weiteren Messungen nicht mehr verfolgt.

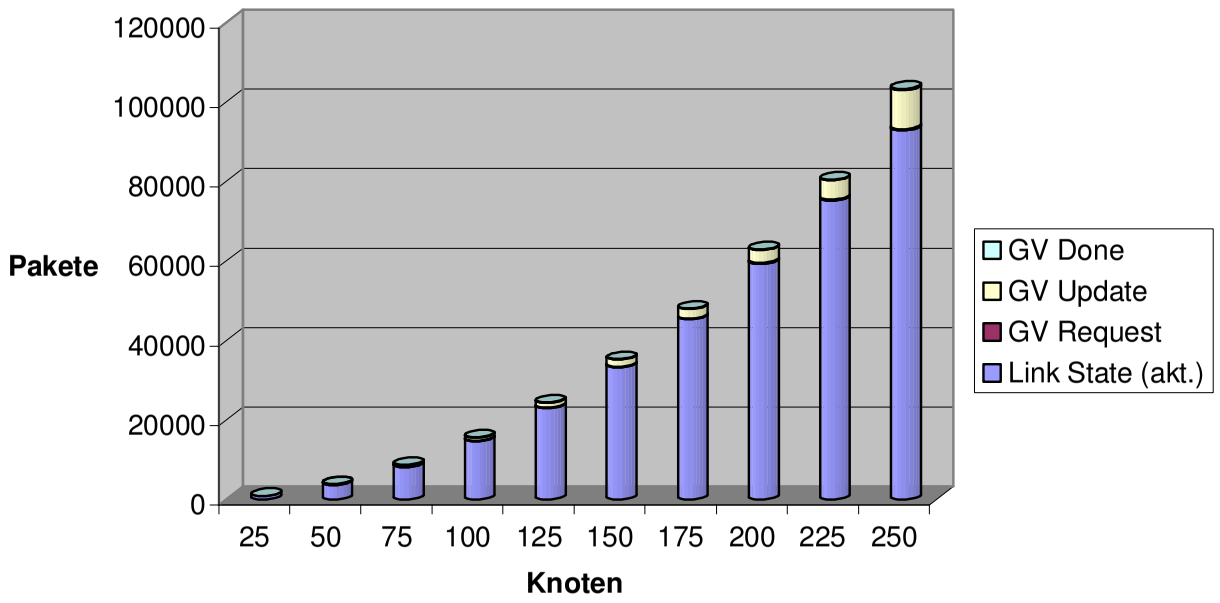
Szenario 4: Knoten – Kosten

In diesem Fall wird untersucht wie viel Nachrichten durch das Hinzufügen eines weiteren Knoten verschickt werden müssen. Dabei werden nicht nur die Nachrichten zum Aufbau des Netzwerks berücksichtigt, sondern ebenfalls die Nachrichten die für einen Schlüsselaufbau zu diesem Knoten. Dadurch gleichen sich Protokolle aus, die entweder den Netzaufbau oder die Pfadsuche optimiert haben.

7.2 Evaluation

Die Durchführung der Testszenarien erfolgte unter Verwendung des G-SPAN Simulators. Die Bereitstellung der Server Dienste (Anmeldung und Generierung der Klienten) erfolgte auf einem 1,3 GHz Intel Pentium M (1 GB RAM, 32 KB L1 Cache, 1024 KB L2 Cache) unter Suse Linux 9.1. Als Compiler wurde der GNU C Compiler in der Version 3.3.3 verwendet. Die Steuerung der Klienten durch die Administrationsoberfläche sowie die Statistikerhebung lief auf einem 3,0 GHz Intel Pentium IV (1 GB RAM, 32 KB L1 Cache, 1024 KB L2 Cache) unter Microsoft Windows XP mit Service Pack 2. Bei dem Compiler für dieses Programm wurde Microsoft Visual Basic 6.0 mit Service Pack 6 verwendet. Die Vernetzung der Rechner erfolgte über ein 100 Mbit Netzwerk.

G-SPAN – Testszenario 1 (Setup)



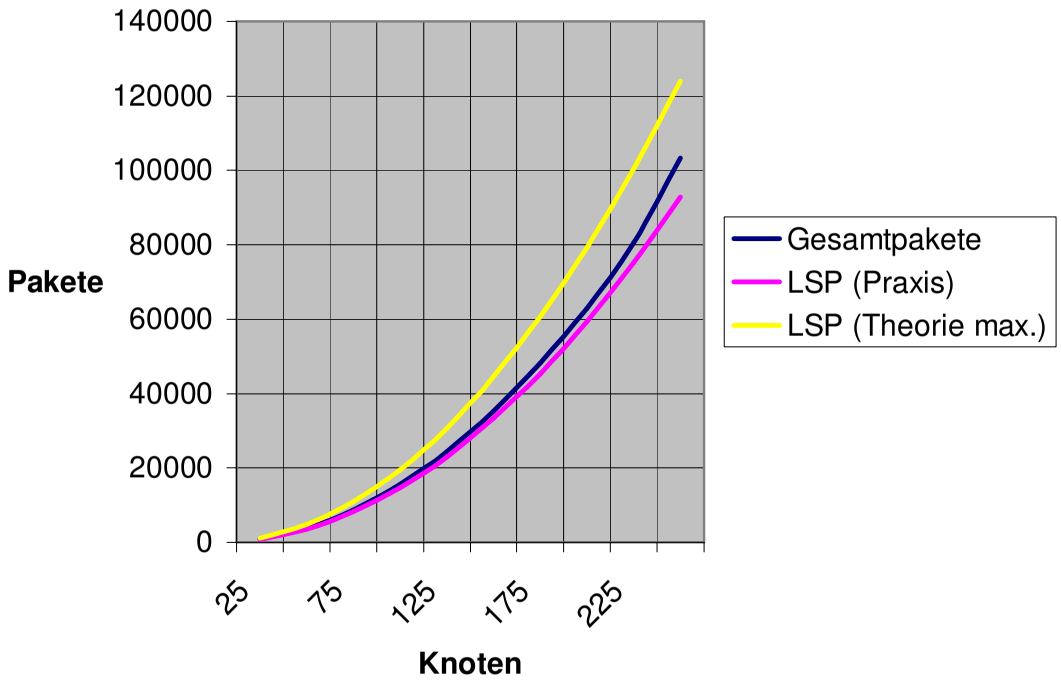
Protokoll:
Maxflow (GV)

Z=2

Bemerkung:

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
max. LSP:	1150	4800	10950	19600	30750	44400	60550	79200	100350	124000
tatsächliche LSP:	851	3576	8176	14651	23001	33226	45324	59299	75148	92834
Abweichung LSP	74,00%	74,50%	74,67%	74,75%	74,80%	74,83%	74,85%	74,87%	74,89%	74,87%
max. GV_REQ:	23	48	73	98	123	148	173	198	223	248
tatsächliche GV_REQ:	23	48	73	98	123	148	174	201	248	374
Verlust an GV_REQs	0	0	0	0	0	0	1	3	25	126

G-SPAN – Testszenario 1 (Setup)



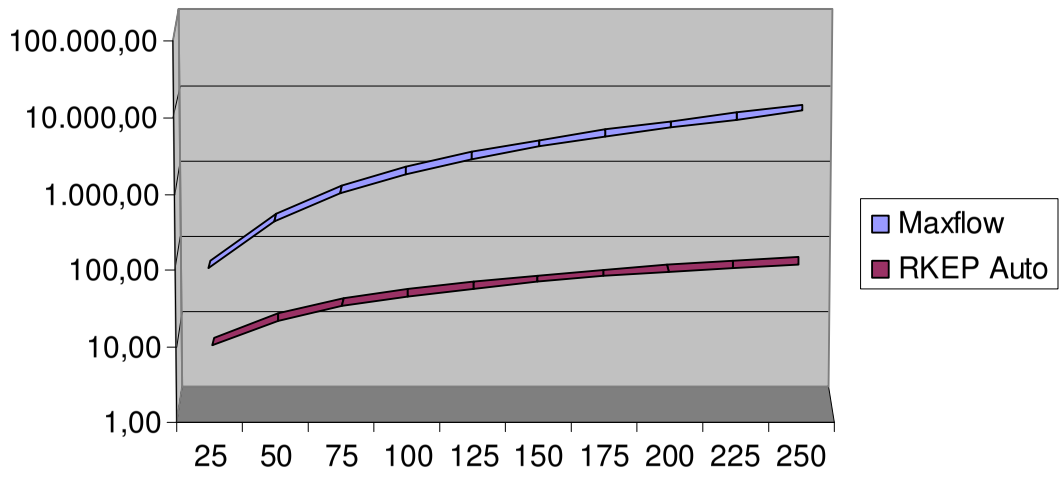
Protokoll:
Maxflow (GV)

Z=2

Bemerkung:
In der Praxis werden ~ 75% aller möglichen LSPs verschickt.

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
max. LSP:	1150	4800	10950	19600	30750	44400	60550	79200	100350	124000
tatsächliche LSP:	851	3576	8176	14651	23001	33226	45324	59299	75148	92834
Gesamtpaketanzahl	952	3879	8777	15646	24486	35298	48095	62932	80482	103329

G-SPAN – Testszenario 1 (Setup)



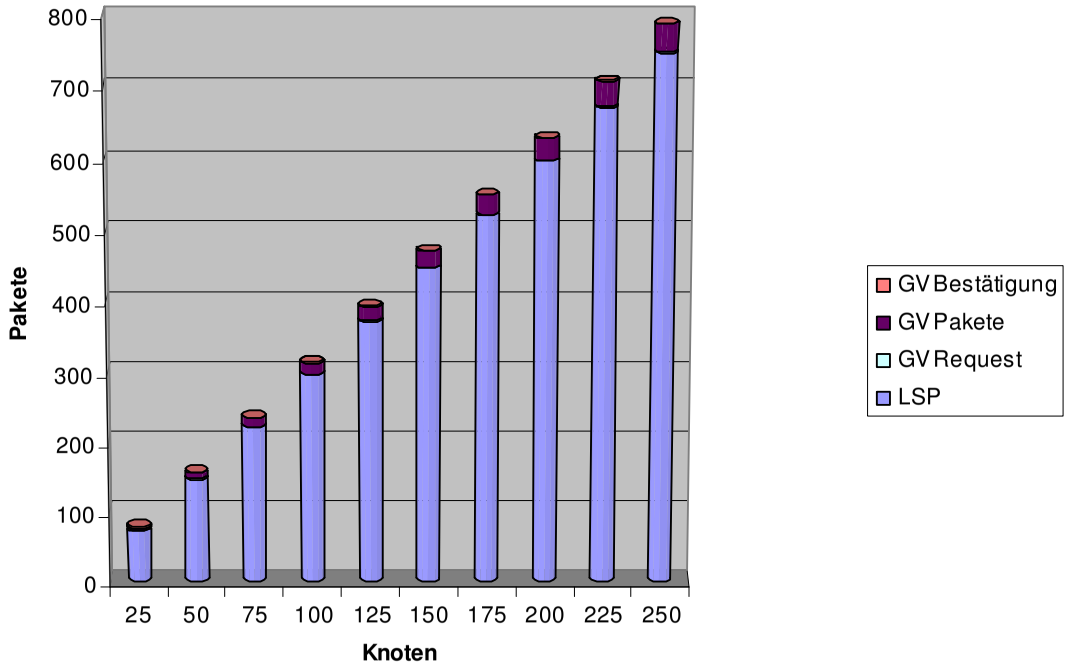
Protokoll:
Maxflow (GV)
RKEP (automatisch)

Z=2

Bemerkung:
Messung des während des Netzwerkaufbaus anfallenden Datenverkehrs in KB.

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
Gesamtpaketanzahl MF	96.655	401.091	913.205	1.632.997	2.560.467	3.695.708	5.039.775	6.597.835	8.430.006	10.768.567
Gesamttransfer MF (in KB)	94,39	391,69	891,80	1.594,72	2.500,46	3.609,09	4.921,66	6.443,20	8.232,43	10.516,18
Gesamtpaketanzahl	156	340	528	712	904	1.100	1.296	1.492	1.688	1.880
Gesamt (in KB)	5,10	11,12	17,27	23,29	29,57	35,99	42,40	48,81	55,22	61,50

G-SPAN – Testszenario 2 (Ein neuer Knoten)



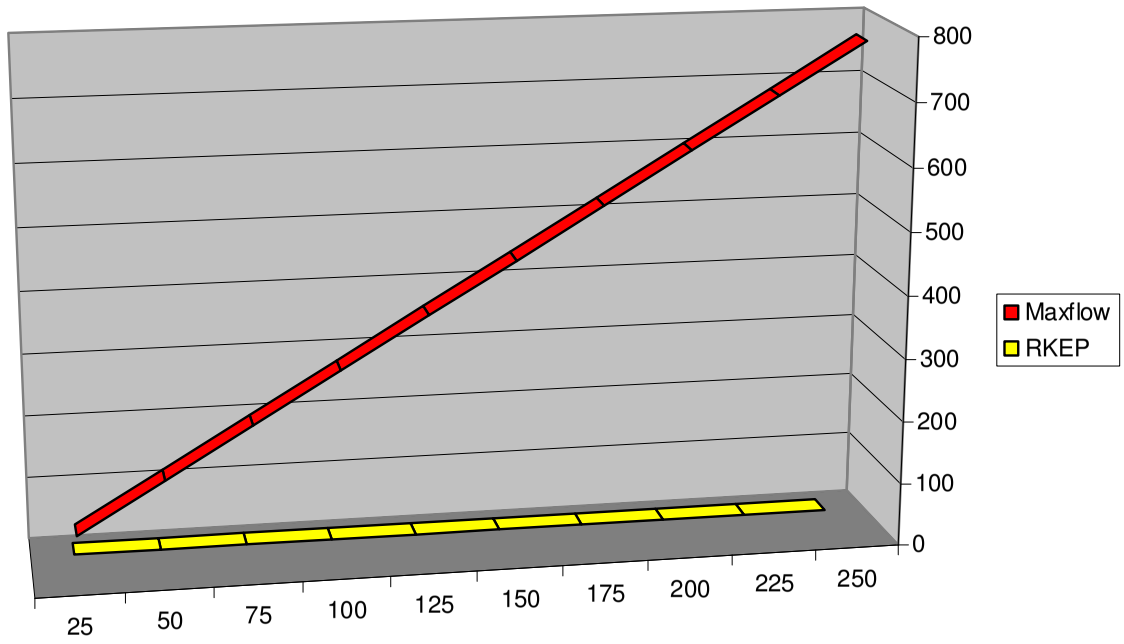
Protokoll:
Maxflow (GV)

Z=2

Bemerkung:
Verteilung der Pakete beim Hinzufügen eines neuen Knotens.

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
LSP	73	148	223	298	373	448	523	598	673	748
GV_REQ	1	1	1	1	1	1	1	1	1	1
GV_UPD	4	8	12	16	20	23	27	31	35	39
GV_DONE	1	1	1	1	1	1	1	1	1	1

G-SPAN – Testszenario 2 (Ein neuer Knoten)



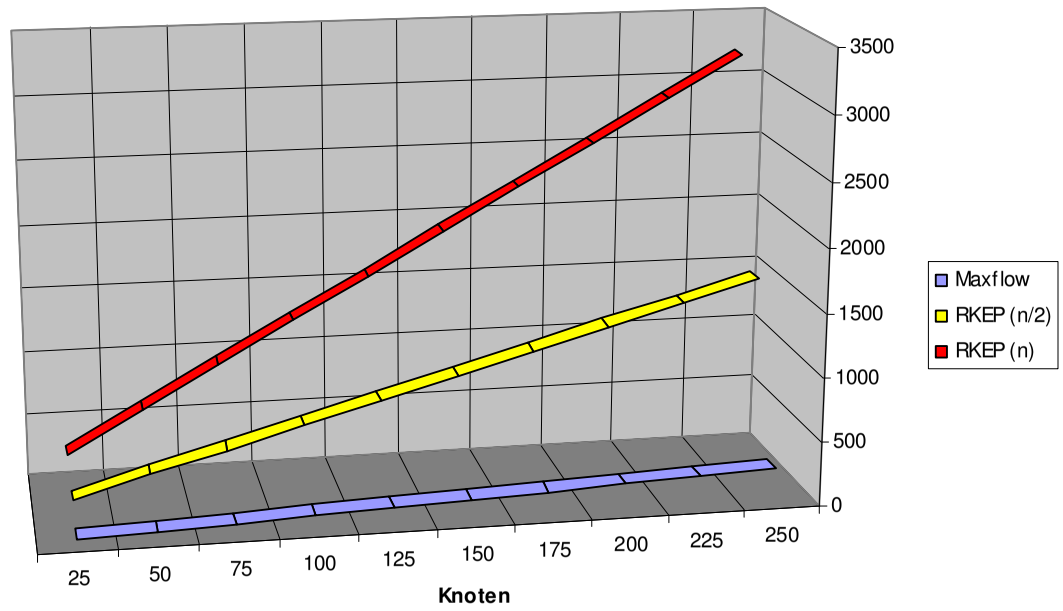
Protokoll:
Maxflow (GV)
RKEP (automatisch)

Z=2

Bemerkung:
Maxflow verschickt LSP und GV Pakete;
RKEP verschickt Nachbarlisten

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
Summe Maxflow	79	158	237	316	395	473	552	631	710	789
Summe RKEP (auto)	2	2	2	2	2	2	2	2	2	2

G-SPAN – Testszenario 3 (Schlüsselaufbau)



Protokoll:
Maxflow (GV)
RKEP (automatisch)
Tiefensuche (iterativ)

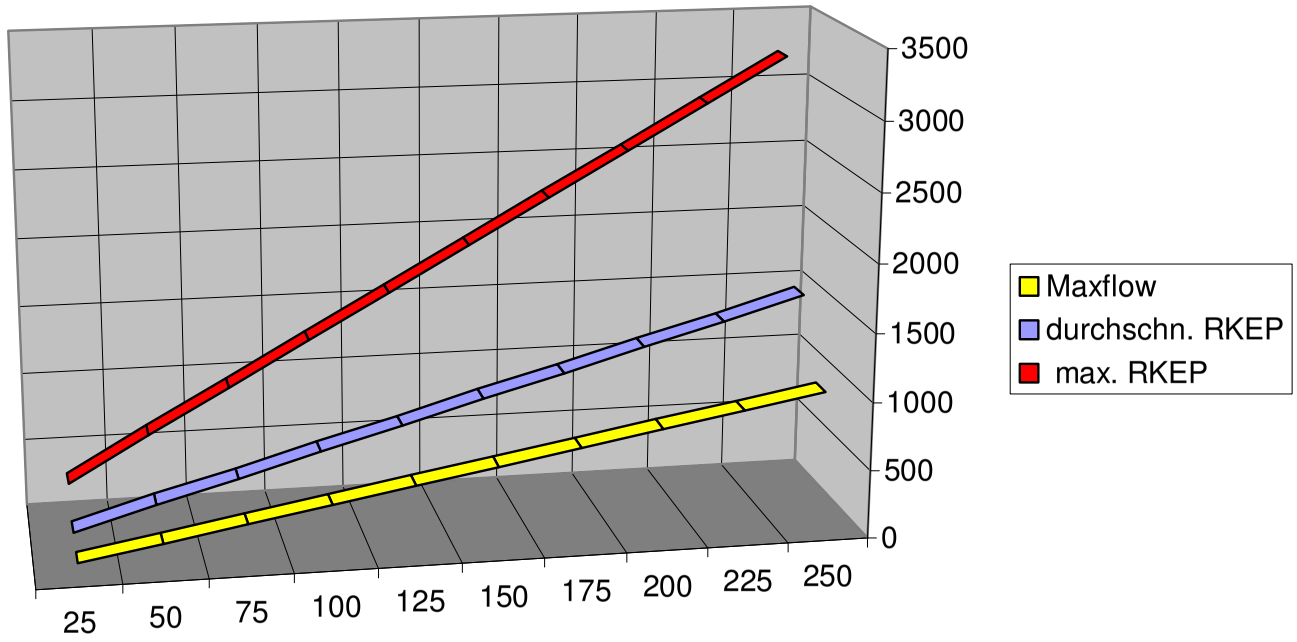
Z=2

Bemerkung:
Die Tiefensuche ist nicht eingezeichnet, da die Paketanzahl zu hoch ist.

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
Gesamtpakete										
RKEP (n/2)	117	286	442	611	767	936	1092	1261	1417	1586
RKEP (n)	278	611	936	1261	1586	1911	2236	2561	2886	3211
Tiefensuche	20649	66649*	81379*							
Maxflow	25	50	75	100	125	150	175	200	225	250

*Pfadsuche nicht erfolgreich

G-SPAN – Testszenario 4 (Gesamtkosten)



Protokoll:
Maxflow (GV)
RKEP (automatisch)

Z=2

Bemerkung:

Knotenanzahl:	25	50	75	100	125	150	175	200	225	250
Kantenanzahl:	47	97	147	197	247	297	347	397	447	497
durschn. RKEP (auto)	119	288	444	613	769	938	1094	1263	1419	1588
max. RKEP (auto)	280	613	938	1263	1588	1913	2238	2563	2888	3213
Maxflow (GV)	104	208	312	416	520	623	727	831	935	1039

8 Resümee

Dieses Kapitel fasst noch einmal alle Ergebnisse dieser Arbeit zusammen. Dabei werden die Vor- und Nachteile der untersuchten Protokolle wiedergegeben und miteinander verglichen. Im Anschluss daran wird noch auf weitere interessante Themengebiete verwiesen. Diese sind aber so umfangreich, dass sie den Rahmen dieser Arbeit sprengen würden und in separaten Arbeiten untersucht werden müssen.

8.1 Zusammenfassung

Die theoretische Analyse sowie die Untersuchung der Protokolle anhand der Testszenarien haben Aufschluss über die Tauglichkeit der Protokolle im G-SPAN Szenario erbracht. Die Szenarioanalyse hat den Bereich, in dem die Algorithmen bezüglich des Nachrichtenaufkommens, sowie ihrem Speicherverbrauch untersucht werden, eingegrenzt.

Es hat sich gezeigt, dass die Maxflow Varianten einen enormen Datenverkehr während der Initialisierung des Netzes hervorrufen. Dieser ist zum Teil bedingt durch die hohe Fehlertoleranz des Protokolls. Verlorene Daten werden nachträglich unter den Knoten synchronisiert, bis alle Knoten dieselbe Sicht auf das Netzwerk haben. Der Aufbau dieser Sicht durch die Link State Pakete benötigt allerdings in jedem Knoten viel Speicher. Allein die Graphtransformation benötigt ein Vielfaches des in den Sensoren zur Verfügung stehenden Speichers. Die Implementierung dieses Protokolls bedarf daher teurerer Sensoren mit größerem Speicher. Der hohe Nachrichtenverkehr wirkt sich bezüglich des Schlüsselaufbaus sehr positiv aus. Nachdem das Netz einmal aufgesetzt wurde, werden für jede Pfadsuche nur maximal n Pakete verschickt. Verglichen mit den anderen Protokollen ist dies der niedrigste Wert zum Schlüsselaufbau. In diesen Paketen muss jedoch der vollständige Pfad von Quelle zu Senke gespeichert sein. Daher begrenzt die Paketgröße die Entfernung, die zwei Knoten maximal voneinander entfernt sein können. Für den Einsatz in Ressourcen-schwachen Geräten eignet sich das Maxflow Protokoll aufgrund seiner Speicheranforderungen und seines hohen Nachrichtenaufkommens in der Anfangsphase daher nicht.

Die Tiefensuch-Protokolle eignen sich ebenfalls nicht für den Einsatz in G-SPAN. Die lokale Tiefensuche scheidet durch das periodische Versenden von Nachrichten aus Energieeffizienzgründen aus. Die Batterien der Geräte würde durchweg beansprucht werden und würden ständiger Kontrolle bedürfen. Die traditionelle Tiefensuche versendet ihre Anfrage-Pakete mit einer Resttiefe entsprechend der maximalen Entfernung zweier Knoten im Netzwerk. Dadurch entstehen ein extremes

Nachrichtenaufkommen und ein hoher Speicherbedarf in den Knoten. Für den Fall, dass diese Faktoren nicht berücksichtigt werden müssten, so ist der Zeitaufwand für einen Schlüsselaufbau aufgrund dieser Nachrichtenanzahl immer noch nicht tragbar. Die iterative Tiefensuche hat einen beschränkten Datenverkehr für den Fall, dass die Geräte, die einen Schlüssel aufbauen wollen, keine große Entfernung aufweisen. Schon ab einer Größe von über 25 Geräten kann die Anzahl der Pakete 20,000 überschreiten. Aufgrund dieser Tatsache scheidet die Tiefensuche für das Heimautomatisierungs-Szenario aus.

Um energieeffizient zu sein, muss die Anzahl verschickter Pakete gering gehalten werden. Dies gelingt nur mit verteilten Verfahren wie RKEP. Dieses Protokoll benötigt zwar eine bestimmte Netzwerkstruktur, die Kosten der Nachbarpakete sind jedoch nicht so hoch wie z.B. bei den Maxflow Protokollen. Des Weiteren können die Nachbarpakete durch Verwendung eines externen Steuergerätes komplett entfallen. Pakete fallen wirklich nur an, wenn eine Pfadsuche bzw. ein Schlüsselaufbau notwendig ist. Die Anzahl der dazu verschickten Pakete ist zwar höher als bei Maxflow, jedoch bezogen auf die Lebenszeit des Netzwerkes deutlich geringer. Abgesehen vom Zielknoten, der viele Schlüssel aufbaut haben die übrigen Knoten fast keinen Speicheraufwand.

Damit erfüllt RKEP als einziges Protokoll die Anforderungen an ein Protokoll für den Einsatz in dem hier untersuchten Szenario mit Ressourcen-schwachen Geräten.

8.2 Ausblick

In dieser Arbeit wurden Aspekte der Sicherheit (s-verbundene Graphen) und Energieeffizienz verschiedener Protokolle untersucht. Dabei bleiben jedoch einige ebenfalls interessante Ansätze aus Platz- und Zeitgründen außen vor.

Durch die Anzahl der Nachrichten, die ein Protokoll sendet lassen sich ungefähre Aussagen über die benötigte Energie während des Betriebs machen. Es bleibt jedoch zu untersuchen, ob es von Vorteil ist den Redundanzfaktor r zu erhöhen, so dass es Backup-Pfade gibt. Senden die Knoten nun mit einer Bake (*Beacon*) [45;50] ihr jeweiliges Energieniveau an die Nachbarknoten, so kann sich ein Teil der Geräte im Netzwerk auf einen Schlummer (*Standby*) Modus schalten, in dem extrem wenig Energie verbraucht wird. Erfolgt eine Kommunikation, bei der ein energiearmer Knoten Pakete weiterleiten soll, haben die anderen Knoten die Möglichkeit, auf die Backup-Pfade auszuweichen (vgl. [51]) und die Funktionalität des Netzes damit über längere Zeit zu garantieren.

Im Zuge dessen kann eine Aufschlüsselung des Energieverbrauchs der einzelnen Operationen (Schlüsselaufbau, Löschen eines Schlüssels aus dem Schlüsselspeicher…) weitere Informationen über die Energieeffizienz der Protokolle liefern. Des Weiteren bleibt zu untersuchen, ob es förderlicher ist eine hohe Anzahl kleiner Pakete oder eine geringe Anzahl größerer Pakete zu verschicken.

Ein weiterer Punkt betrifft die Verschlüsselung im Netzwerk. Bisher wurden aus Gründen mangelnder Rechenkapazität auf den Einsatz von asymmetrischer Verschlüsselung verzichtet. Durch die Entwicklung und Verwendung neuartiger Verfahren, wie die elliptische Kurvenkryptographie, ist es mittlerweile möglich geworden auch auf Ressourcen-schwachen Geräten (siehe [5]) asymmetrische Verschlüsselung zu verwenden. Der Umstieg auf ein asymmetrisches Verfahren in G-SPAN würde dazu führen, dass sämtliche Protokolle umgeschrieben werden müssen. Es bleibt zu untersuchen, ob der Aufbau dadurch vereinfacht werden kann und gegebenenfalls Synergieeffekte auftreten.

Mit dem Wachhund (*Watchdog*) [8;52] stehen weitere Ansätze zur Verfügung die Sicherheit im Netzwerk zu erhöhen. Es wird davon ausgegangen, dass die Funkreichweite jedes Knotens seine 2-Hop Nachbarn mit einschließt. Schickt ein Knoten (A) einem Knoten (B) ein Paket, welches dieser noch weiterleiten muss, so wechselt A in einen promiskuitiven Zustand und hört das Netzwerk ab. Dadurch kann A überprüfen ob B das Paket weitergeschickt hat. Ist dies nicht der Fall, so handelt es sich bei B um einen fehlerhaften Knoten, der umgangen werden muss. Ein Pfadbewerter (*Pathrater*) übernimmt diese Aufgabe und sucht Pfade zwischen Start- und Zielknoten, ohne den fehlerhaften Knoten zu verwenden. Mit dieser Implementierung könnte die Angreifererkennung (*Intrusion Detection System*) ausgebaut werden.

Ein Problem mit dem der G-SPAN Simulator zu kämpfen hat, sind die Paketverluste (verursacht z.B. durch ein erhöhtes Paketaufkommen bei großer Knotenanzahl oder einer DOS Attacke) durch den Einsatz von UDP. Der Vorteil von UDP in Sensornetzwerken liegt zum einem in der Einfachheit der Kommunikation (kein Verbindungsaufbau) und der optimalen Nutzung der Ressourcen (Energie) durch variable Paketgrößen. Dadurch, dass aber keine Bestätigung der Pakete stattfindet, kommt es häufig zu wiederholtem Senden von Paketen (*Retransmits*). Dies kann oftmals nur durch Timeouts realisiert werden, was zusätzlich zur Verlangsamung des Systems führt. Die naheliegendste Lösung stellt die Umstellung auf einen bestätigten Dienst dar, der eine bezüglich der Funktionalität zwischen UDP und TCP liegt. Eine Evaluation kann zeigen, ob der Ausbau von UDP (→ *Reliable UDP*) gegenüber einer reduzierten Version von TCP die effektivere Lösung darstellt.

Die Implementierung einer Last-Verteilung bringt weitere Vorteile um die Ressourcen der Knoten zu schonen. Dadurch wird eine gleichmäßigere Verteilung der Anzahl der Schlüsselpartner pro Knoten erreicht. Dies hat zur Folge, dass weniger Schlüssel aufgrund eines drohenden Überlaufs des Schlüsselspeichers gelöscht werden müssen. Im Maxflow Protokoll erhöht diese Erweiterung die Robustheit des Verfahrens. Durch eine gleichmäßige Schlüsselverteilung steigt die Anzahl der Pfade zwischen den Knoten. Dadurch entstehen weitere Backup-Pfade, so dass sogar Knotenausfälle toleriert werden können. In diesem Fall wird einfach der Backup-Pfad gewählt und der Teilschlüssel über diesen verschickt. Da

Alternativpfade meist länger sind als die primär gefundenen Pfade muss hier ein Abstrich in der Performanz (vgl. [53]) des Protokolls durch die entstehende Verzögerung zugunsten der Robustheit hingenommen werden.

A Die Administrations-Software

Zur Visualisierung und zur Durchführung von Messungen wird auf das Programm **secAdmin** (Version 2.8 siehe [12]) zurückgegriffen. Dieses Programm bietet Funktionen, die zur Evaluation von Algorithmen in Bezug auf die Kenngrößen wie Anzahl bestimmter Nachrichtentypen herangezogen werden kann.

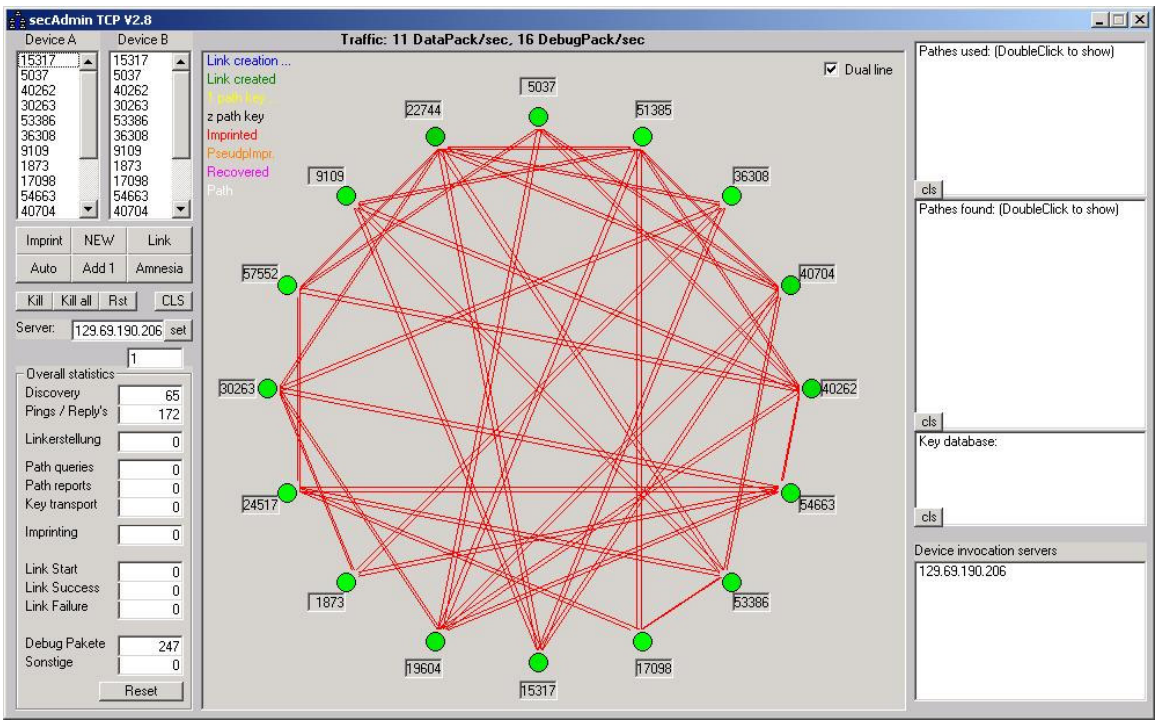


Abbildung 20 - SecAdmin V2.8

Grundfunktionen

Der folgende Abschnitt stellt eine Übersicht über die wichtigsten Funktionen dar, die es dem Anwender ermöglichen, ein Netzwerk aus Sensoren am Computer zu simulieren.

Visualisierung:

Der Oberflächenprototyp visualisiert die Sensoren durch Knoten und das Netzwerk durch Linien zwischen diesen Knoten. Diese Verbindungen stehen für asynchrone Schlüssel, die sich die Schlüsselpartner teilen, um miteinander abhörsicher kommunizieren zu können. Wurde die Verbindung korrekt aufgebaut, so existiert

zwischen den Partnern eine Doppellinie. Dies bedeutet, dass beide Partner über einen Schlüssel zur Kommunikation mit dem anderen verfügen.

Verbindungstypen:

Dabei wird zwischen zwei Typen von Verbindungen unterschieden. Rot gekennzeichnete Linien entsprechen imprinteten Schlüsselpaaren, schwarz gekennzeichnete Linien stellen Schlüsselpaare dar, die durch eine Pfadsuche aufgebaut wurden.

Interaktionsmöglichkeiten:

Des Weiteren bietet das Programm die Möglichkeit, weitere Knoten zum Netzwerk hinzuzufügen und diese mit ausgewählten Partnern zu imprinten. Dadurch hat der Benutzer die Möglichkeit, sein „real-life“ Szenario im Rechner nachzubilden und zu testen. Wenn Sensoren miteinander kommunizieren wollen, aber nicht über einen gemeinsamen Schlüssel verfügen, so wird die Pfadsuche initiiert. Die Länge der gefundenen Pfade, die zum Schlüsselaufbau herangezogen werden, ermöglichen es verschiedene Netztopologien miteinander zu vergleichen und so im Durchschnitt möglichst kurze Pfade zwischen häufig miteinander kommunizierenden Knoten zu erlauben.

Statistikinformationen:

Die Oberfläche bietet weiterhin die Möglichkeit, statistische Daten zu erfassen. Dazu gehören eine feste Anzahl von Nachrichtentypen und einige zusätzliche Informationen:

- Anzahl der Discovery Nachrichten
- Anzahl der Ping / Pong Nachrichten
- Anzahl der Nachrichten zur Linkerstellung
- Anzahl der Anfragen zur Pfadsuche
- Anzahl und Wegwahl der gefundenen Pfade
- Anzahl der Pakete mit Schlüsseln
- Anzahl imprinteter Verbindungen
- Anzahl gestarteter, erfolgreicher und nicht erfolgreicher Pfadsuchen
- Anzahl sonstiger Pakete

Erweiterung

Die Messsoftware hatte bis zu Beginn dieser Arbeit nur den Status eines Prototyps erreicht. Einfache Funktionen, die dem Anwender grundlegende Informationen liefern, wurden zwar schon realisiert, jedoch sind diese Informationen teilweise recht ungenau und nicht ausreichend. Weitergehende Funktionen zum Vergleich der verschiedenen Protokolle stehen aber nicht zur Verfügung. Daher wurde das Programm auf der bestehenden Basis unter Microsoft Visual Basic V6.0 [54] weiterentwickelt. Auf die einzelnen neuen Funktionen wird im folgenden Abschnitt genauer eingegangen:

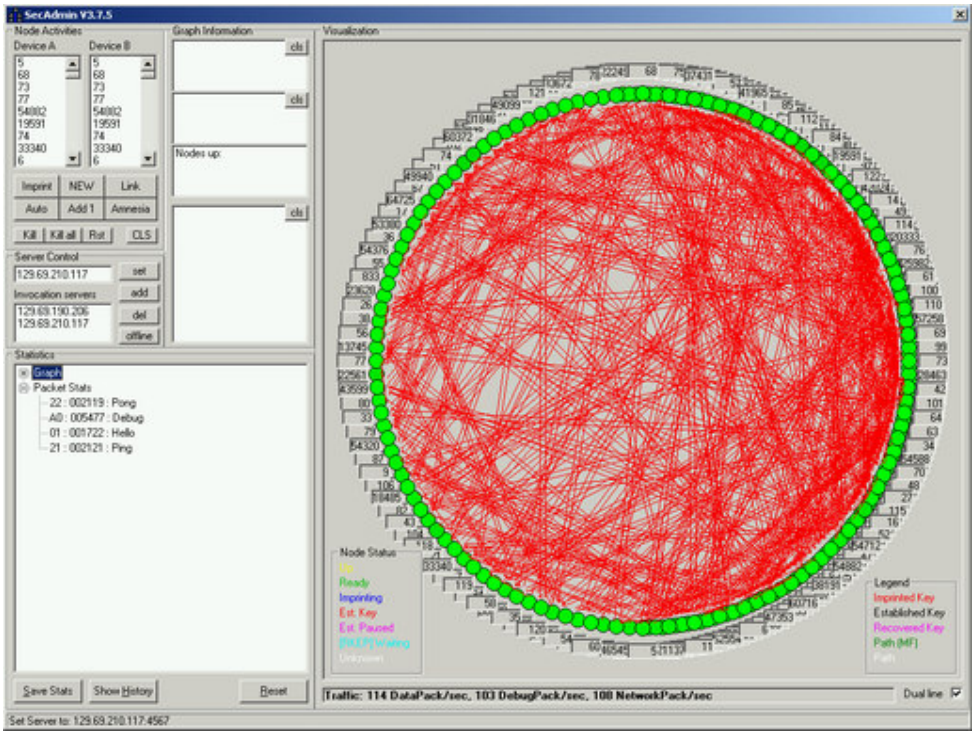


Abbildung 21 - SecAdmin V3.7.5

Knoteninformationen:

Die Oberfläche visualisiert die Sensoren durch farbige Knoten, welche den Zustand derer widerspiegeln. Der Initialzustand jedes Knoten ist *up* und wird durch eine gelbe Füllung symbolisiert. Sobald der Schlüsselspeicher des Sensors z überschreitet wechselt der Zustand auf *ready*, dargestellt durch eine grüne Füllung. Jeder Eintrag im Schlüsselspeicher stellt einen Schlüssel und somit eine sichere Kommunikationsmöglichkeit zu einem anderen Sensor dar. Die Darstellung erfolgt über rote (imprintete) und schwarze (über Pfadsuche aufgebaute) Verbindungslinien.

Paketstatistik:

Bisher war die Paketstatistik statisch aufgebaut, das heißt es standen vordefinierte Felder zur Verfügung, die die Anzahl bestimmter Pakettypen anzeigen. Dadurch ist es nicht möglich Nachrichtentypen anderer Protokolle zu überwachen und gegeneinander zu vergleichen.

Diese Felder wurden durch eine dynamische Baumstruktur ersetzt, die alle auftretenden Pakettypen überwacht und zählt. Zu Informationszwecken wird – soweit in der Typ Datenbank (siehe nächster Abschnitt) hinterlegt – noch eine Paketbeschreibung mit angezeigt um die Bedeutung der Nachrichtentypen darzulegen. Durch diese Erweiterung lassen sich auch alle später entwickelten Protokolle auf ihr Nachrichtenverhalten untersuchen und mit bisher bekannten in Bezug bringen.

Datenbankanbindung:

Zur Erhebung von Statistiken musste bisher die Anzahl der verschiedenen Pakettypen mühsam aus den einzelnen Feldern übernommen bzw. herauskopiert werden.

Daher wurde das Programm um eine Speicherfunktion erweitert. Die Anzahl sämtlicher Pakettypen sowie der wichtigsten Kennzahlen des Graphen (Knoten- und Kantenanzahl sowie z) werden nun unter Verwendung von ActiveX Bibliotheken in einer Tabelle einer Microsoft Access [54] gespeichert. In einer weiteren Tabelle werden alle bisher bekannten Pakettypen mit ihrer Beschreibung gelistet. Dies ermöglicht eine schnelle und effiziente Weiterverarbeitung (Vergleiche, Diagrammerstellung…) durch andere Softwareprodukte wie beispielsweise Microsoft Excel [54].

Verlauf:

Dies ist eine der wichtigsten Neuerungen. Die derzeitige Darstellungsform der Knoten auf der Kreisfläche sowie eine mit jedem Aufbau andere Graphstruktur haben bisher einwandfreie Vergleiche erschwert beziehungsweise sogar verhindert. Bisher wurden die Knoten IDs bei jedem Graphaufbau (= Netzwerkaufbau) zufällig vergeben. Bei der Pfadsuche war es daher nicht möglich, die Anzahl der Knoten zwischen Start und Ziel (*intermediate hops*) zu bestimmen und sie somit mit der Paketanzahl in Verbindung zu bringen. Dies ist aber sehr wichtig, da verschiedene Protokolle nur am selben Graphen akkurat verglichen werden können. Vergleiche anhand von Durchschnittswerten sind daher in diesem Fall kaum verwendbar.

Nun wird im Verlauf während des Graphaufbaus in chronologischer Reihenfolge gespeichert, welcher Knoten wann mit welchem Partner eine Verbindung aufbaut.

Diese können in einer Datei auf einem Datenträger gespeichert werden und für spätere Analysen wieder verwendet werden. Nach Laden der Verlaufsdatei wird der exakte selbe Graph rekonstruiert. Danach kann die Pfadsuche zwischen ausgewählten Knoten unter Verwendung der verschiedenen Algorithmen wieder und wieder gestartet werden und im Paketstatus-Fenster die Anzahl der benötigten Pakete zum Verbindungsaufbau beobachtet werden. Dies ermöglicht eine genauere Aussage über die Effizienz beim Nachrichtenversand.

Konfigurationsdatei:

Bisher wurden einige wichtige Parameter wie zum Beispiel der Standard-Server zur Entwurfszeit fest in der Software verankert. Andere Kennzahlen wie z konnten erst bei fortgeschrittener Netzwerkgröße durch den Empfang bestimmter Pakettypen bestimmt werden.

Die Einführung einer Konfigurationsdatei erlaubt es nun, diese Werte direkt auszulesen. Gleichzeitig kann eine Liste verfügbarer Server mitgegeben werden, die zur Laufzeit beliebig erweitert werden kann. Die Unterteilung der Datei in Sektionen ermöglicht eine problemlose Erweiterung um zusätzliche Kennzahlen für weitere Protokolle.

B Referenzen

[1] LAN Pro AG. Universelle Gebäudeverkabelung. URL: www.lanpro.ch, (20-10-2004), 2005

[2] 3PC - Peer to Peer Pervasive Computing. URL: www.3pc.info, (3-9-2004), 2005

[3] Sascha Faber, "Digital Home" in *PC Professionell,* Ziff Davies, München, Deutschland, 2004, S. 170-178.

[4] Alfred Menezes, Paul van Oorschot, and Scott Vanstone, *Handbook of Applied Cryptology,* 5th ed. CRC Press, Boca Raton, FL, USA 1996.

[5] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes, "PGP in Contrained Wireless Devices" in *9th Usenix Security Symposium,* Denver, CO, USA, 2000 S. 247-261.

[6] "Specification for the Advanced Encryption Standard" Federal Information Processing Standards Publication (FIPS PUB), Nr. 197, 2001.

[7] Stefano Basagni, Kris Herrin, Danilo Bruschi, and Emilia Rosti, "Secure Pebblenets" in *ACM MobiHOC,* Long Beach, CA, USA, 2001.

[8] Sergio Marti, T.J.Giuli, Kevin Lai, and Mary Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks" in *6th ACM/IEEE International Conference on Mobile Computing and Networking,* 2000 S. 255-265.

[9] Lidong Zhou and Zygmunt J.Haas, "Securing Ad Hoc Networks" in *IEEE Network Magazine,* 13. ed 1999, S. 24-30.

[10] Arno Wacker, Timo Heiber, Holger Cermann, and Pedro José Marrón, "A Fault-tolerant Key-Distribution Scheme for Securing Wireless Ad-hoc Networks" in *2nd International Conference on Pervasive Computing,* Linz/Wien, Österreich, 2004.

[11] Arno Wacker, Timo Heiber, and Holger Cermann, "A Key-Distribution Scheme for Wireless Home Automation Networks" in *1st IEEE International Consumer Communications and Networking Conference,* Las Vegas, NV, USA, 2004.

[12] Holger Cermann, "Design, prototypische Implementierung und Evaluierung einer Sicherheitsschicht für ressourcenbeschränkte Geräte." Universität Stuttgart, 2003.

[13] Teco Smart-ITs. URL: http://smart-its.teco.edu/, (20-2-2005), 2005

[14] Ramesh Bhandari, "Optimal Physical diversity algorithms and survivable networks" in *2nd IEEE Symposium on Computers and Communications,* Alexandria, Egypt, 1997 S. 433-441.

[15] Mahesh K.Marina and Samir R.Das, "On-demand Multipath Distance Vector Routing in Ad Hoc Networks" in *International Conference for Network Protocols,* Riverside, USA, 2001.

[16] Subhash Suri, "Network Flows", URL: http://www.cs.ucsb.edu/~suri/cs230/NetworkFlows.pdf, 2004.

[17] Stavros G.Kolliopoulos and Clifford Stein, "Approximating Disjoint-Path Problems Using Greedy Algorithms and Packing Integer Programs" in *IPCO,* 1998.

[18] Dahai Xu, Yang Chen, Yizhi Xiong, Chunming Qiao, and Xin He, "On Finding Disjoint Paths in Single and Dual Link Cost Networks" in *IEEE Infocom,* 2004.

[19] Heike Rippenhausen-Lipa, Dorothea Wagner, and Karsten Weihe, "The Vertex-Disjoint Menger Problem in Planar Graphs" in *SIAM Journal on Discrete Mathematics*, Vol. 26, Nr. 2, S. 331-349, 1997.

[20] Home Automation Inc. URL: www.homeauto.com, (20-10-2004), 2005

[21] Sensor Technik. URL: www.wikipedia.de, (1-3-2005), 2005

[22] Marcel Jacomet, "Analogtechnik" Fachhochschule Biel, Schweiz, 2003.

[23] Micrsoft Corporation. Easy Living. URL: http://research.microsoft.com/easyliving, (22-10-2004), 2005

[24] Conrad Haustechnik. URL: www.conrad.de/haustechnik, (20-10-2004), 2005

[25] 3S-design GmbH. 3S Xccess Zugangskontroll-System. URL: http://zugang.3sdesign.de, (22-10-2004), 2004

[26] A.N.Tolstoi, "Methods of Finding Minimum Total Kilometrage in Cargo Transportation Planning in Space" in *Transportation Planning,* Nr. 1, 1930.

[27] Alexander Schrijver, "On the history of transportation and maximum flow problems" in *Mathematical Programming*, Vol. Series B91, S. 437-445, 2001.

[28] Thomas Hofmeister, "Operations Research" Lehrstuhl Informatik 2, Universität Dortmund, 2001.

[29] William J.Cook, William H.Cunningham, William R.Pulleyblank, and Alexander Schrijver, *Combinatorial Optimization*. John Wiley & Sons, Inc 1998, S. 38-126.

[30] Andreas Brandstädt, *Graphen und Algorithmen*. Teubner: Stuttgart, Deutschland, 1994, S. 124-148.

[31] Ravindra K.Ahuja, Thomas L.Magnati, and James B.Orlin, *Network Flows: Theory, Algorithms and Applications*. Prentice Hall: Englewood Cliffs, New Jersey, USA, 1993, S. 166-255.

[32] Sariel Har-Peled, "Network Flows II - The Vengeance" University of Illinois, Champaign, IL, USA, 2003.

[33] Boris V.Cherkassky and Andrew V.Goldberg, "On Implementing Push-Relabel Method for the Maximum Flow Problem" in *Algorithmica*, Vol. 19, Nr. 4, S. 390-410, 1997.

[34] Andrew V.Goldberg, Boris V.Cherkassky, and Joao C.Setubal, "Augment or Push ? A computational study of Bipartite Matching and Unit Capacity Flow Algorithms" NEC Research Institute, Inc., Nr. #98-036R, 1998.

[35] Andrew V.Goldberg and Robert E.Tarjan, "A New Approach to the Maximun Flow Problem" in *18th ACM Symposium on Theory of Computing,* Berkeley, CA, USA, S. 136-146.

[36] V.King, S.Rao, and Robert E.Tarjan, "A Faster Deterministic Maximum Flow Algorithm" in *3rd Sympsium on Discrete Algorithms,* 1992 S. 157-164.

[37] Yuri Boykov and Vladimir Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision" in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 26, Nr. 9, S. 1124-1137, 2004.

[38] David R.Karger and C.Stein, "A new approach to the minimum cut problem" in *Journal of the Acm*, Vol. 43, Nr. 4, S. 601-640, 1996.

[39] David R.Karger, "Random sampling in cut, flow, and network design problems" in *26th ACM Symposium on Theory of Computing,* Montreal, Quebec, Canada, S. 648-657.

[40] David R.Karger, "On Finding Disjoint Paths in Single and Dual Link Cost Networks" in *29th ACM Symposium on Theory of Computing,* El Paso, TX, USA, S. 240-249.

[41] Kevin Wayne, "COS226 Algorithms & Data Structure: Max Flow, Min Cut".

[42] Hermann Engesser, *Duden Informatik,* 2nd ed. Dudenverlag (Brockhaus AG): Mannheim, Deutschland, 1993.

[43] Arno Wacker, "A New Approach for Establishing Pairwise Keys for Securing Wireless Ad-hoc Networks" 2005.

[44] Dr.Susanne Wetzel. Battery Draining and Node Killing Strategies. Stevens Institute of Technology, Hoboken, NJ, USA, 6-8-2004

[45] Patrick Kinney, "ZigBee Technology: Wireless Control that Simply Works" in *Communication Design Conference,* 2003.

[46] Internet Engineering Task Force (IETF), "RFC 1583" 1994.

[47] Radia Perlman, *Interconnections,* 2nd ed. Addison Wesley: Redwood City, CA, USA, 1999.

[48] Yu Liu and David Tipper, "Successive Survivable Routing for Node Failures" in *Globecom,* San Antonio, TX, USA, 2001.

[49] Sung-Ju Lee and Mario Gerla, "Split Multipath Routing with Maximally Disjoint Paths in Ad hoc Networks" in *IEEE International Conference on Communications,* Helsinki, Finland, 2001 S. 3201-3205.

[50] William C.Craig, "ZigBee: "Wireless Control that Simply Works"" ZMD America, Inc., 2005.

[51] Anand Srinivas and Eytan Modiano, "Minimum Energy Disjoint Path Routing in Wireless Ad-hoc Networks" in *ACM MobiCom,* San Diego, CA, USA, 2003.

[52] Jean-Pierre Hubaux, Levente Buttyán, and Srdan Capkun, "The Quest for Security in Mobile Ad Hoc Networks" in *ACM MobiHOC,* Long Beach, CA, USA, 2001.

[53] Asis Nasipuri, Robert Castaneda, and Samir R.Das, "Performance of Multipath Routing for On-Demand Protocols in Ad Hoc Networks (formerly: On-demand multi-path routing for mobile ad-hoc networks)" in *ACM/Kluwer Mobile Networks and Applications (MONET) Journal*, Vol. 6, Nr. 4, S. 339-349, 2000.

[54] Microsoft Corporation. URL: www.microsoft.com, (12-1-2005), 2005

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst und nur die angegeben Quellen verwendet zu haben.

_____

Mirko Knoll