fhwedel
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Wedel

Seminararbeit WS2004/2005

Thema:

Lokale Variante des Algorithmus von Dijkstra zur Berechnung kürzester Wege und das Problem der Aktualisierung von Wegen bei Leitungsausfällen

Eingereicht von: Alex Prentki (wi4512)
alex@prentki.de

Erarbeitet im: 9. Semester

Abgegeben am: 14.12.2004

Referent: Prof. Dr. Sebastian Iwanowski
Fachhochschule Wedel
Feldstraße 143
22880 Wedel

Inhaltsverzeichnis

1. Einleitung

1.1. Problemstellung

Das Finden kürzester Wege zwischen zwei Punkten ist in vielen Bereichen der Wissenschaft von großer Bedeutung. Dabei ist die Berechnung dieser Wege nicht trivial, da in den interessanten Fällen Start- und Zielpunkt nicht direkt mit einander verbunden sind. Der klassische Fall ist das Logistikproblem: Hier gilt es, die Transportwege in einem Distributionssystem optimal zu gestalten.
Aber auch viele Probleme im Bereich der Künstlichen Intelligenz können als Suchprobleme dargestellt werden. Der Suchraum kann dabei durch einen gerichteten Graphen dargestellt werden und in vielen Fällen interessiert dann der kürzeste Weg einer Startecke zu einer bestimmten Zielecke. Beispiele dafür wären Strategiespiele wie Schach oder Dame. Um einen möglichst guten Schachzug durchführen zu können, muss man alle Züge, die der Gegner als nächstes machen kann, bedenken. Weiterhin möchte man möglichst schnell (auf kürzestem Wege) zu einer Zielstellung gelangen. Das Schachbeispiel zeigt ein weiteres Problem bei der Suche nach kürzesten Wegen: Die Anzahl der Knoten im Suchraum ist häufig sehr groß. Nimmt man an, dass bei einer gegebenen Stellung 25 Züge möglich sind und man fünf eigene und fünf gegnerische Züge vorausplanen will, so muss man 25^{13} Züge überschauen![1]
Weitere Probleme treten auf, wenn es zu Leitungsausfällen in Kommunikations-netzen kommt. Da der Algorithmus dort bezüglich der Laufzeiteffizienz an seine Grenzen stößt, ist es vorstellbar, für solche Situationen etwas neues zu entwickeln.

1.2. Zielsetzung

Ziel dieser Arbeit soll es sein, dem Leser die Funktionsweise und den Nutzen des Dijkstra Algorithmus verständlich zu machen. Weiterhin soll auf seine, besonders im Routing, praktische Relevanz aufmerksam gemacht werden. Dazu muss dieses Werk ein grundlegendes Verständnis der Routingtechnik vermitteln. Ein weiteres Ziel ist es, sich der Problematik von Leitungsausfällen in Computernetzwerken anzunehmen und Lösungsvorschläge dafür zu entwickeln, wie Router mittels des in dieser Arbeit vorgestellten Algorithmus damit umgehen könnten.

[1] vgl. Turau, Volker S.256-257 „Kürzeste-Wege-Probleme in der künstlichen Intelligenz“

1.3. Gang der Untersuchung

Zu Beginn des zweiten Kapitels wird zunächst der Erfinder des Algorithmus vorgestellt. Anschließend werden in Abschnitt 2.2 die notwendigen Grundlagen der Graphentheorie erläutert, um damit dann in Abschnitt 2.3 den eigentlich Algorithmus detailliert zu erklären.
Kapitel 3 befasst sich mit dem Problem der Aktualisierung beim Ausfall von Leitungswegen. Es beginnt in Abschnitt 3.1 mit einer Einführung in die Thematik des Routing. Dort werden Technik, Verfahren und Realisierung des Routings beschrieben. Der Abschnitt 3.2 befasst sich anschließend mit den Auswirkungen von Leitungsausfällen in Routingsystemen. Abschließend werden mit Abschnitt 3.3 Lösungsvorschläge entwickelt, diskutiert und bewertet.
Die Arbeit endet mit Kapitel 4 mit einem Schlußwort des Verfassers.

2. Der Dijkstra Algorithmus

2.1. Edsger Wybe Dijkstra

Herr E.W. Dijkstra (1930-2002), geboren in Rotterdam, war ein international angesehener Computerwissenschaftler. Er lebte in Holland und arbeitete lange an der Eindhoven Universität für Technologie. Anfang der 70er Jahre war er in der Forschungsabteilung der „Burroughs Corporation“, einem Hardware Unternehmen für Schreibmaschinen und Computer, tätig. 1972 erhielt Herr Dijkstra den nach dem Erfinder der Turing Maschine bennanten „Turing Award“, der auch der Nobelpreis der Computerwissenschaft genannt wird. Die Zeit von 1984 bis 2000 verbrachte er in den Vereinigten Staaten von Amerika und hatte dort den „Schlumberger Centennial Chair in Computer Sciences“ an der „University of Texas“ in Austin inne. Zu seinen herausragenden Leistungen gehörte die Entwicklung des Algorithmus zur Berechnung kürzester Wege.[2]

2.2. Grundlagen aus der Graphentheorie

Zum besseren Verständnis des Dijkstra Algorithmus sind Erläuterungen zu einigen wichtigen Begriffen aus der algorithmischen Graphentheorie hilfreich. Daher werden im Folgenden grundlegende Definitionen ausführlicher erklärt.

[2] vgl. University of Texas http://www.cs.utexas.edu/users/EWD/

Ein **ungerichteter Graph** besteht aus einer Menge E von **Ecken** und einer Menge K von **Kanten**. Eine Kante ist gegeben durch die Verbindung eines ungeordneten Paares von Ecken (einfacher: den Endpunkten der Kante). Die Ecken werden oft auch Knoten oder Punkte genannt. „Ungerichtet“ heißt es deswegen, weil zwischen Anfangs- und Endpunkt der Kante nicht unterschieden wird.

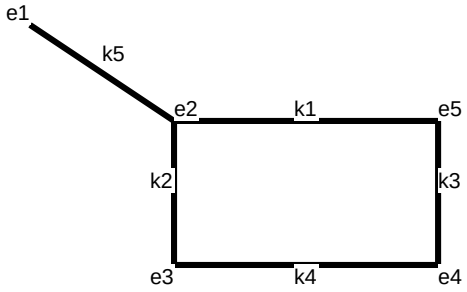

Abbildung 1: ungerichteter Graph

Von **Nachbarn** spricht man, wenn zwei Ecken e und f eines ungerichteten Graphen benachbart sind, d.h. wenn es eine Kante von e nach f gibt. In Abbildung 1 sind die Ecken e_1, e_3 und e_5 Nachbarn von e_2.

Ein **Pfad** von einem Anfangsknoten a zu einem Endknoten z ist eine Knotenfolge (oder auch Kantenfolge) von p Knoten $e_1,e_2,...,e_p$ mit e_1=a, e_p=z. Die Knotenfolge kann einen Knoten mehrfach durchlaufen (Wiederholungsmöglichkeit). Ein **Weg** von einem Anfangsknoten a zu einem Endknoten z ist ein Pfad bei dem alle verwendeten Kanten verschieden sind, also ohne Wiederholungsmöglichkeit. Ist der Startknoten gleich dem Endknoten des Weges, so heißt der Weg **geschlossen**, andernfalls heißt er **offen**.

Bei **bewerteten Graphen** wird jeder Kante ein Wert aus der praktischen Anwendung zugewiesen. Abbildung 2 zeigt einen solchen Graphen. Die Bewertungen spiegeln in diesem Beispiel die Entfernungen zwischen den Städten in Kilometern wider. In einem bewerteten Graphen ist die **Länge eines Weges** zwischen einem Startknoten s und einem Zielknoten z gegeben als die Summe der Bewertungen auf den Kanten des Weges.

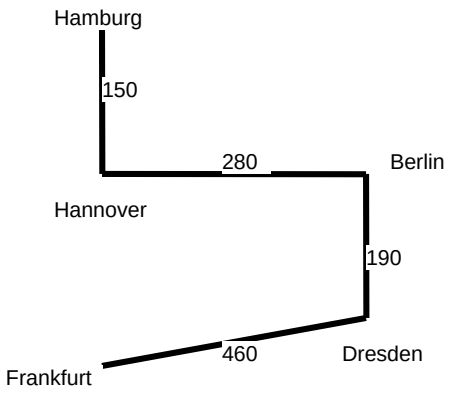

Abbildung 2: bewerteter Graph

Der Dijkstra Algorithmus erfordert die Darstellung der Punkte in einem **kürzesten-Wege-Baum** (kW-Baum). Dieser kW-Baum ist immer ein Untergraph eines

kantenbewerteten Graphen G, bei dem alle Kantenbewertungen positiv bzw. ≥ 0 sind. Gibt es keine Kante von einem Punkt i zu einem Punkt j, so ist die Bewertung dafür auf B[i,j] = ∞ zu setzen. Der kW-baum ist ein gerichteter Baum mit der Wurzel s. Ferner ist die Eckmenge E′ des kW-Baumes gleich der Menge der Ecken des Graphen G, welche von der Wurzel s aus erreichbar sind. In so einem kW-Baum gilt dann, dass für jede Ecke e **c** E′ der eindeutige Weg von der Wurzel s nach e ein kürzester Weg von s nach e in G ist.[3]

2.3. Der Algorithmus

Der Dijkstra Algorithmus basiert auf einer geschickten Auswahl der zu bearbeitenden Ecken, so dass jede Ecke nur einmal betrachtet werden muss. Bei jedem Bearbeitungsschritt wird der Vorgänger der ausgewählten Ecke, über den diese Ecke *zum gegenwärtigen Zeitpunkt* auf kürzestem Wege zu erreichen ist, gespeichert. Zur Auswahl der Ecken wird dabei eine Menge B (bekannt) von Ecken verwaltet, für welche schon ein Weg von der Startecke (Wurzel) aus bekannt ist. Eine Ecke i ist in B, wenn die Distanz D[i] zur Wurzel ≠ ∞ ist. Die Distanzen aller Ecken, exkl. der unmittelbaren Nachfolger der Wurzel, werden anfangs mit ∞ initialisiert. Die Distanz der Ecken in B werden in jedem Schritt mit ihrer *zum Zeitpunkt des Bearbeitungsschrittes* kürzesten Distanz zur Wurzel überschrieben. Diese Distanz entspricht der Summe der Kantenbewertungen aller Kanten des bisher kürzesten Weges zu dieser Ecke. Die Menge B verändert sich daher mit dem schrittweisen Aufbau des kW-Baumes. Die Wege zu diesen Ecken sind aber noch nicht unbedingt die kürzesten Wege! Bei jedem Schritt wird eine Ecke aus B entfernt und bearbeitet. Aus der Menge B wird immer die Ecke mit der kleinsten Distanz D[i] zur nächsten Bearbeitung ausgewählt. Gibt es mehrere Ecken mit minimaler Distanz, kann davon eine beliebige genommen werden. Die Bearbeitung selbst beinhaltet die Speicherung des Vorgängers über den diese Ecke auf dem tatsächlich kürzesten Wege zu erreichen ist. Wurde eine Ecke aus der Menge B entfernt und bearbeitet, so kommt sie in die Menge E (endgültig markiert) und muss nie wieder berücksichtigt werden. Die weiteren Überprüfungen beschränken sich daher auf alle Nachfolger j **c** B. Wird nur der kürzeste Weg von der Wurzel zu einer Ecke z gesucht, kann das Dijkstra

[3] Definitionen aus Vahrenkamp, Dr. Richard Kapitel 2.3 „Gerichtete Netzwerke“ und 2.5 „Bewertete Netzwerke“, sowie Turau, Volker Kapitel 2.1 „Grundlegende Definitionen“

Verfahren abgebrochen werden, sobald z die Menge B verlässt und in die Menge E der endgültig markierten Knoten aufgenommen wird. Im ungünstigsten Fall enthält die Menge E am Ende alle Ecken des Graphen G, d.h. es wurden alle Knoten betrachtet.

Da das Einfügen und Entfernen von Ecken aus B den gleichen Zeitaufwand erfordert und jede Ecke genau einmal bearbeitet wird, ergibt sich eine Laufzeit von $O[n^2]$, wobei n die Anzahl der Knoten ist.[4]

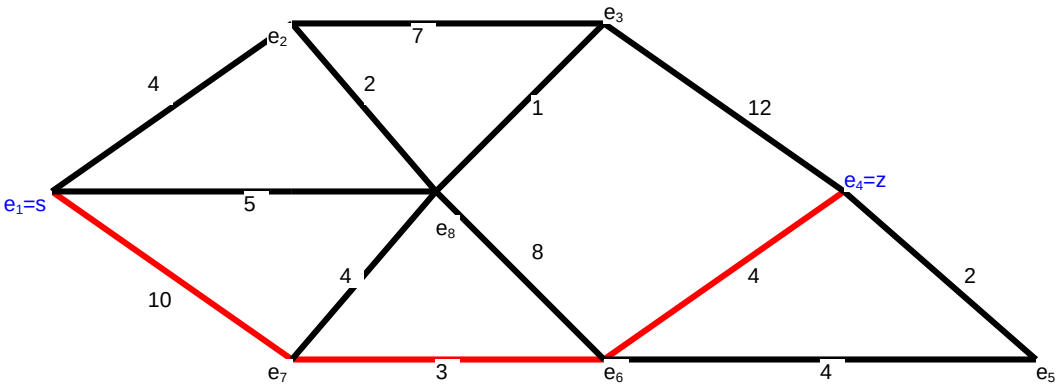

Abbildung 3: kW-Baum

3. Aktualisierung von Wegen bei Leitungsausfällen

3.1. Routing

3.1.1. Begriffsdefinitionen

Dieser Abschnitt beschreibt die wichtigsten Begriffe aus der der Welt des Routing.

Routing	Der Prozess, der dafür dafür sorgt, dass ein Datenpaket in einem IP Netzwerk an das korrekte Ziel weitergeleitet wird.
Router **Host** **Server**	Ein Hardwaregerät, das Protokolle auf dem IP Layer interpretieren kann und die Datenpakete auf einem Pfad in Richtung ihres korrekten Endziels weiterleitet. In der Graphentheorie entspricht der Router einem Knoten.
Gateway	Entspricht einem Router, der das Koppelelement zwischen

[4] vgl. Turau, Volker Kapitel 8.4 „Anwendung auf spezielle Graphen“ Seite 250-254

	zwei physisch voneinander getrennten Netzwerken ist.
Routing Deamon	Ein auf dem Server laufender Prozess, der die Routingtabelle verwaltet.
Routing Algorithmus	Datenpakete in Computernetzwerken werden nach diesem Algorithmus verarbeitet: 1. Zieladresse des Datenpaketes untersuchen 2. Adresse in Netzwerk- und Hostanteil aufteilen 3. Ermitteln, ob Host im selben Netzwerk a. wenn ja, direkt an diesen weiterleiten b. wenn nein, Paket anhand der Routingtabelle an den nächsten Router weiterleiten[5]

Tabelle 1: Begriffsdefinitionen im Routing

3.1.2. Routingverfahren

In Kommunikationsnetzen, wie z.B. dem Fernsprechnetz, sind die (Telefon-) Stationen normalerweise während des gesamten Datenaustausches direkt miteinander verbunden. Computernetzwerke, wie z.B. das Internet, verwenden dagegen sogenannte *Routingverfahren*. Dabei sind Sender und Empfänger nicht unmittelbar miteinander verbunden, sondern die Datenpakete gelangen stationsweise von Router zu Router. Der Sender übergibt dem Netzwerk seine mit der Adresse des Empfängers versehene Nachricht, welches diese dann selbstständig übermittelt. Die Nachricht wird dabei in Fragmente aufgeteilt, die anschließend unabhängig voneinander übertragen werden. Die Aufgabe eines Routers ist es dann, einen möglichst günstigen Weg für die Datenpakete durch das Netz zu finden. Jede Station kennt dabei nur ihre direkten Nachbarn und weiß zusätzlich, welcher Nachbar für ein bestimmtes Ziel ausgewählt werden muss.

[5] Begriffsdefinitionen aus IBM Redbooks Kapitel 4.3. „Routing Terminology“ Seite 74

Routingverfahren können generell in zwei Gruppen aufgeteilt werden, die sich dann jeweils noch weiter differenzieren:

1) **Zentralisierte Verfahren**
   Zentrale Verwaltung von Routinginformationen.
2) **Verteilte Verfahren**
   Im Netz verteilte Routinginformationen.
   a) **Statische Routingverfahren**
      Feste Wegwahl durch manuelle Definition einer Routingtabelle.
   b) **Dynamische Routingverfahren**
      „Ständige“ Aktualisierung der Routingtabelle.
      - **Isolierte Verfahren**
        Jeder Router sammelt seine eigenen Routinginformationen, ohne Austausch mit den Nachbarn.
      - **Verteilte Verfahren**
        Router tauschen Informationen untereinander aus.[6]

Das *Statische Routing* hat den Nachteil, dass es anfällig gegenüber Leitungsausfällen ist und auch keine unterschiedlichen Netzbelastungen berücksichtigen kann. Daher wollen wir uns im Folgenden auf der Ebene der *Verteilten Verfahren* mit dem *Dynamischen Routing* auseinandersetzen.

3.1.3. Dynamisches Routing

Beim dynamischen Routingverfahren werden Eigenschaften jeder Verbindung periodisch gemessen. Als Messgröße können dabei verschiedene Parameter dienen. Wird z.B. die „durchschnittliche Verzögerung“ genommen, so entspricht diese der Zeitspanne, die ein Paket in einer Station verweilen muss, bevor es weitergeleitet wird. Das Netzwerk wird dabei als gerichteter Graph dargestellt, auf dem die Kantenbewertungen dem Parameter entsprechen. Jede Station verwaltet seine eigene Version dieses Graphen.
Das Verfahren heißt *dynamisch*, weil jede Station in periodischen Zeitabständen die Parameter der Verbindungen zu seinen direkten Nachbarn über das Netzwerk versendet und die Router so gegenseitig Informationen austauschen. Jeder Router,

[6] vgl. Kaleck, I. „Rechnernetze für die Wirtschaftsinformatik und die Medieninformatik“ Seite 372

der neue Informationen über die Verbindung erhält, aktualisiert damit sofort seinen Graphen. Bei dem Verfahren ist allerdings darauf zu achten, dass dieser Datenaustausch im Verhältnis zur gesamten Netzwerkkapazität gering bleibt.
Um nun unter Berücksichtigung dieser Parameter den kürzesten Weg zu finden, kann auch hier der Algorithmus von Dijkstra angewendet werden. Die Berechnung erfolgt dezentral in jedem Router, so dass die Abhängigkeit von einem Zentral-rechner entfällt. Jede Station berechnet für jedes mögliche Ziel nach dem Dijkstra Algorithmus ihren kW-Baum. Dadurch kennt jeder Knoten die gesamte Netzwerktopologie, speichert für jedes Ziel jedoch nicht den vollständigen Weg, sondern nur den ersten Knoten auf diesem Weg ab. Diese Daten werden in der *Routingtabelle* verwaltet.[7]

Wenn Ziel Router	2	3	4	5	6	7
Dann weiterleiten an	2	3	3	2	4	3

Abbildung 4: Routingtabelle

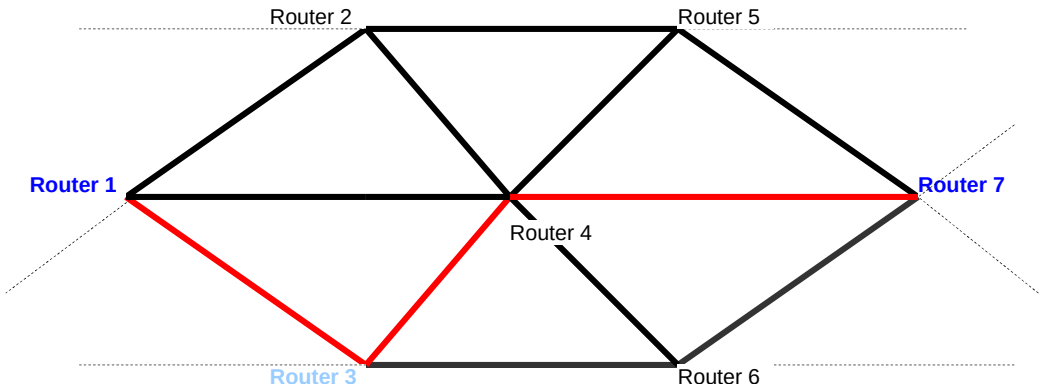

Abbildung 5: Dynamisches Routing als kW-Baum

Es gibt verschiedene Protokolle, die das Dynamische Routing auf der Hardwareebende umsetzen. Die zwei verbreitetesten sind *RIP* und *OSPF* und werden in den nächsten beiden Abschnitten kurz vorgestellt.

[7] vgl. Turau, Volker Kapitel 8.5 „Routingverfahren in Kommunikationsnetzen“ Seite 255-256

3.1.4. Routing Information Protocol (RIP)

Das RIP benutzt den *hop count* (auch *distance vector*), um den bestmöglichen Pfad zu einem Host oder einem anderen Netzwerk zu finden. RIP wird häufig auch als *distance vector Verfahren* bezeichnet. Der hop count ist eine Art Entfernungs- oder Kostenangabe, die angibt, wie schnell ein bestimmter Server zu erreichen ist. Alle Netzwerke, die direkt mit dem Router verbunden sind, haben einen hop count von null, alle die über genau ein gateway erreicht werden können, erhalten einen hop count von eins, usw. Dabei gilt, je weniger desto besser. Der Pfad mit den wenigsten hop counts zum tatsächlichen Ziel wird bevorzugt. Ein hop count von sechzehn bedeutet, dass das Ziel unendlich weit entfernt ist oder das es nicht erreichbar ist. Diese Entfernungsangabe wird auch als *Metrik* des Pfades bezeichnet. Durch die Beschränkung auf maximal fünfzehn Gateways ist RIP in sehr großen Netzwerken nicht mehr einsetzbar.
Die hop count Informationen stehen in der *distance vector Tabelle*, einer erweiterten Routingtabelle, und werden in regelmäßigen Zeitabständen, wie in Kapitel 3.1.4. „Dynamisches Routing“ beschrieben, mit allen benachbarten Router ausgetauscht. Empfängt ein Router drei Minuten lang keine Aktualisierungsinformationen von einem anderen, so wird dieser Router als nicht mehr erreichbar markiert und alle Pfade über ihn erhalten einen hop count von sechzehn. Tabelle 1 zeigt die Inhalte einer solchen Tabelle.[8]

Adress	Die IP Adresse des Zielhosts oder –netzwerks
Subnet Mask Value	Erweiterung von RIP 2 zur besseren Benutzung von dynamisch hinzugefügten Pfaden.
Gateway	Das erste Gateway auf dem Pfad zum Zielhost.
Interface	Angabe, welches physische Netzwerk benutzt werden muss, um das erste Gateway erreichen zu können.
Metric	Zahlenangabe über die Entfernung zum Ziel.
Timer	Zeit, die seit der letzten Aktualisierung vergangen ist.

Tabelle 2: Inhalte einer distance vector (Routing)Tabelle

[8] vgl. IBM Redbooks Kapitel 4.8 „Routing Information Protocol (RIP)“ Seite 89

3.1.5. Open Shortest Path First (OSPF)

Beim OSPF Protokoll enthält die Routingtabelle mehrere Details über die Art der Verbindungen zu den einzelnen Routern. Es werden Informationen über den Status (aktiv oder inaktiv) und über die „Kosten“ des Weges (Netzbelastung, Bandbreite, hop count, etc.) gespeichert. Dieses *least-cost-Prinzip* ermöglicht es, Pfade auf Grund verschiedenster Eigenschaften zu bewerten. Somit lassen sich die realen Bedingungen der Netzwerktopologie differenzierter abbilden und man gelangt zu besseren Ergebnissen bei der Berechnung des tatsächlich kürzesten Weges.[9]

Bei der Aktualisierung mit OSPF findet der Datenaustausch mit anderen Routern, im Gegensatz zum nachbarorientierten RIP, als *broadcast* statt. Dabei wird jedes mal, wenn sich die Parameter einer Verbindung ändern, nicht die gesamte Routingtabelle übertragen, sondern nur die entsprechende Änderung an alle Router gesendet. Dadurch wird die Netzbelastung durch derartige Zusatzinformationen gegenüber RIP reduziert. Ein weiterer Vorteil von OSPF ist es, dass es keine Restriktion der Routing Metrik gibt und das Verfahren dadurch auch für sehr große Netzwerke einsetzbar ist.[9]

OSPF basiert auf dem Konzept des *link-state Routings*. Da jeder Router die Kosten der Verbindung zu seinen direkten Nachbarn kennt, tauscht er diese mit allen anderen Routern über *link-state-advertisements* (LSAs) aus. Dieser Austausch wird der „regelmäßige Austausch“ genannt und findet ca alle 30 Minuten statt. Jeder Router baut dann mit diesen empfangenen Informationen seine *link-state-database* auf. Diese Datenbank beschreibt somit die Topologie des gesamten Netzwerks. Da alle Router eine identische Version dieser Datenbank verwalten, kann jeder eine Änderung in der Netzwerktopologie sofort erkennen. Aus dieser link-state Datenbank wird dann auch mittels des Dijkstra Algorithmus die eigentlich Routingtabelle berechnet.[9]

Da es im Internet sehr sehr viele Router gibt, fasst OSPF logisch zusammengehörige Router in *areas* zusammen und verwaltet jeweils nur innerhalb dieser areas identische Datenbanken. Auch areas tauschen sich miteinander aus. Im Rahmen

dieser Arbeit wollen wir jedoch nicht tiefer in die Spezifikation dieses Protokolls eindringen.[9]

3.2. Problem der Aktualisierung

Da die in den beiden vorhergehenden Kapiteln vorgestellten Routingprotokolle sich ausschließlich auf der Hardwareebene dem Problem eines Leitungsausfalls annehmen, stellt sich nun die Frage nach den Auswirkungen auf die ursprünglich berechneten kürzesten Wege zwischen den Knoten auf der logischen Ebene.
Zu Problemen bei der Zustellung von Datenpaketen in Netzwerken kommt es immer dann, wenn z.B. wegen eines Unwetterschadens, einzelne Leitungsabschnitte temporär ausfallen. Je nach eingesetztem Routingverfahren, erkennen die Router nach einiger Zeit diesen Leitungsausfall und müssen einen neuen kürzesten Weg berechnen. Da der Dijkstra Algorithmus solche Knotenausfälle nicht berücksichtigt, startet nun jeder Router eine neue Berechnung aller kürzesten Wege. Da es im Internet aber unzählige Pfade über sehr sehr viele Router gibt, erscheint es aufwändig, bei jedem Ausfall einer Leitung, alle Wege komplett neu zu berechnen. Tatsächlich ist der Rechenaufwand enorm, wird jedoch durch die Leistung aktueller Rechenprozessoren relativiert. Gibt man sich jedoch nicht mit einer dearartigen Lösung zufrieden, so muss man entweder einen geeigneteren Algorithmus entwickeln oder aber die Ergebnisse des Dijkstra Algorithmus geschickter verwalten. Im folgenden Kapitel möchte ich eigene Ansätze zur letzteren Möglichkeit vorstellen.

3.3. Lösungsansätze

3.3.1. Idee

Als Lösungsstrategie habe ich den Weg vom Abstrakten zum Konkreten gewählt. Daher wollen wir uns zunächst nicht weiter um die technische Realisierung bzw. die genauen Einträge in der Routingtabelle kümmern.
Wir nehmen einfach an, dass ein Router einmal alle kürzesten Wege für jedes Ziel mittels des bekannten Algorithmus bestimmt und abgespeichert hat. Will man nun eine komplette Neuberechnung bei Ausfall eines Knotens vermeiden, müsste man einen Pfad kennen, der lediglich den ausgefallenen Knoten umgeht, bis er auf einen anderen Knoten des Pfades trifft. Danach könnte der ursprüngliche Pfad fortgesetzt

[9] vgl. IBM Redbooks Kapitel „Open Shortest Path First (OSPF)“ Seite 82-89

werden. Die Komplexität dieser zunächst simpel erscheinden Lösung steigt, wenn man die Anzahl der Möglichkeiten betrachtet: Jeder Pfad besteht aus sehr vielen Knoten und es müsste solche Umgehungswege für jeden Pfad und jeweils alle Knoten auf den Pfaden geben! Aber auch diese Erkenntnis soll uns nicht davon abhalten, diese Idee weiter zu verfolgen, da eine derartige Berechnung ja nur einmalig bzw. selten durchgeführt werden müsste.
Doch wie findet ein Router solche Umgehungswege?

3.3.2. Realisierung

Abbildung 6 soll uns im Folgenden als beispielhafte Netzwerktopoogie dienen und den Lösungsvorschlag veranschaulichen.

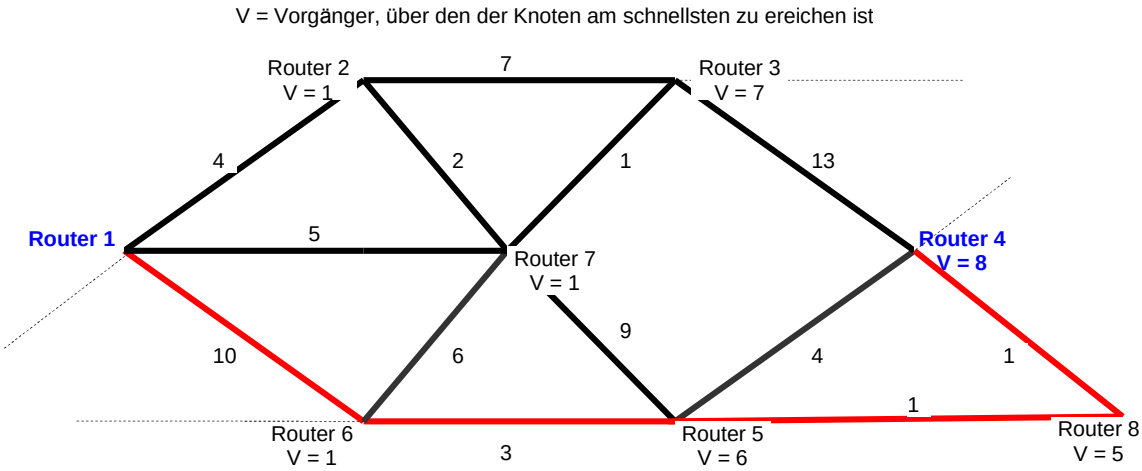

Abbildung 4: Beispiel einer Netzwerktopologie

In unserem Beispiel betrachten wir Router 4 als den Zielknoten und lassen unser Datenpaket bei Router 1 starten. Alle Router haben dafür nach dem Dijkstraverfahren den roten Pfad als den kürzesten berechnet. Dabei kennt natürlich jeder Router nur den ihm nachfolgenden Teil des Pfades.

<u>Beispiel 1</u>

Es kommt nun zu einem Ausfall von Router 6. Router 1 muss nun wissen, dass er den Ausfall über Router 7 umgehen kann und ab Router 5 den ursprünglichen Pfad fortsetzen könnte.

Beispiel 2

Router 5 ist defekt und kann nicht benutzt werden. Sein Vorgänger müsste nun über Router 7 → Router 3 zum Zielknoten 4 gelangen. Dabei braucht er sich nur Router 7 zu merken, da dieser wiederum Nummer 3 als den Richtigen kennt.

Da jeder Router in seiner Routingtabelle für jedes Ziel sowieso nur seinen unmittelbaren Nachbarn eintragen muss, reicht es aus, „nur“ für jedes Ziel und jeden Nachbarn einen Umgehungsweg zu kennen. Jeder Knoten sollte daher für jedes Ziel diesen Weg unter der Bedingung, dass der betroffene Nachbarknoten ausfällt, berechnen. Die Idee dabei ist, dass der Dijkstra Algorithmus bereits abgebrochen wird, wenn ein Knoten des ursprünglichen kürzesten Pfades erreicht wird. Ab dort ist der weitere Pfad ja bereits aus der ursprünglichen Berechnung bekannt.
Voraussetzung dafür ist, dass der ursprüngliche Pfad zusätzlich zur Routingtabelle in einer Datenbank gespeichert wurde. Die Berechnung dieser Umgehungswege könnte man immer dann durchführen, wenn die Netzbelastung gering ist, um nicht den normalen Nutzdatentransport zu beeinträchtigen.
Die brechneten Umgehungswege könnten im Router selbst als Array abgelegt werden: `Umgehungswege[ausgefallenerKnoten] = UmgehungsKnoten`. Im ersten Beispiel würde das Array `Umgehungswege[6] = 7` liefern.

3.3.3. Bewertung

Der Vorteil dieses Ansatzes liegen darin, dass zum Zeitpunkt des Leitungsausfalls keine aufwendige Berechnung mehr durchgeführt werden muss, da ja eine Umgehung bekannt ist. Nachteilig ist dagegen der höhere Datenbestand durch die Abspeicherung aller kompletten kürzesten Pfade und der Umgehungswege. Es muss nun Rechenaufwand (oder Zeit) gegen Speicherkapazität abgewogen werden. Denkbar ist auch ein partieller Einsatz dieses Verfahrens in zeitkritischen Leitungsabschnitten, die keine längeren Ausfallzeiten erlauben.

3.3.4. Anregungen

Bei zukünftigen Verbesserungen des Algorithmus bezüglich dieser Problemstellung, sollte man noch die Eigenschaft des Dijkstra intensiver ausnutzen, dass bei jeder Wegsuche als Nebeneffekt sämtliche kürzesten Wege zu allen bearbeiteten Knoten berechnet wurden! Dadurch könnten die Umegungswege optimiert werden. Das

Problem des jetzigen Vorschlags, dass solche Umgehungspfade teilweise nur partiell optimal sind, wäre damit zu lösen.
Auch das dynamische Hinzuschalten von neuen Servern und die erneute Inbetriebnahme von ausgefallenen Servern, sollten bei der Weiterentwicklung dieses Ansatzes berücksichtigt werden, um eine umfassende Lösung zu erhalten.

4. Schlusswort

Wissenschaftler und Techniker geben sich heute mit dem bisherigen Verfahren nach Dijkstra und der Laufzeit $O[n^2]$ zufrieden und sehen wenig Anlass zur weiteren Forschung auf diesem Gebiet.
Der vorgestellte Lösungsansatz soll auch nicht als neueres, besseres Verfahren verstanden werden, sondern viel mehr als eine zu überdenkende Alternative zum Bisherigen. Dabei ist zu bedenken, dass die vorgestellte Lösung im schlechtesten Fall definitv nicht schneller ist, aber bei Durchschnittsbetrachtungen durchaus besser abschneiden würde! In einem sehr großen Netzwerk, wie z.B. dem Internet, liegt es nahe, dass ein Umgehungspfad schneller berechnet ist, als alle kompletten kürzesten Pfade.
Weiterhin ist diese Arbeit, als ein Einstieg in die Thematik der Wegoptimierung zu sehen. Insgesamt beschränkte auch der zeitliche Umfang dieser Seminararbeit meine Forschungen auf diesem Gebiet.

A. Literaturverzeichnis

Turau, Volker

Algorithmische Graphentheorie, Addison-Wesley 1996, ISBN 3-89319-938-1

IBM Redbooks

Communications Server for z/OS V1R2 TCP/IP Implementation Guide

Volume 4: Connectivity and Routing (englisch), Oktober 2002, Adolfo Rodriguez, Marcia Maria Fascini, Giancarlo Rodolfi, Heather Woods, ISBN 0738424145

Vahrenkamp, Prof. Dr. Richard

Quantitative Logistik für das Supply Chain Management, 2003, R. Oldenbourg Verlag München Wien, ISBN 3-486-27369-8

B. Tabellenverzeichnis

C. Abbildungsverzeichnis