

## Kurzbericht Projekt 11

# OSPF Simulation in Haskell

Reto Lehnherr, Gabriel Zimmerli

### Abstract

Unser Programm berechnet die kürzesten Pfade zwischen Routern gemäss dem OSPF-Protokoll und gibt eine Routing Tabelle aus. Das Programm simuliert den Ausgangsrouter in einem Netzwerk. Die Nachbarschaftstabelle, sowie die Link State Updates (LSU) der anderen Router werden aus einem File eingelesen und geparkt. Der Kürzeste Pfad wird anhand des erstellen Netzwerk-Graphen mit dem Dijkstra-Algorithmus berechnet. Schwerpunkt der Implementation liegt beim Parsen der LSU und dem Dijkstra-Algorithmus.

### Idee des Projekts

Wir möchten einen Teil des OSPF-Protokolls implementieren. Die Idee ist, dass unser Programm einen Router im Netz simuliert. Anhand von einer Tabelle sollen die Nachbarschaften zu anderen Routern deklariert sein.

Von diesen Nachbarn werden dann Link-State-Updates empfangen. Da unser Programm lediglich einen Router darstellt, werden dies Updates Files auf dem Dateisystem sein. Diese Files müssen ausgewertet werden um anschliessend eine Topologie Tabelle erstellen zu können. Danach wird anhand dieser Topologie Tabelle mit Hilfe des Dijkstra Algorithmus der «Shortest Path Tree» und damit schlussendlich die Routing-Tabelle erstellt.

Als Input haben wir ein Text-File mit einer Nachbarschaftstabelle, wie sie der Router nach Erhalt der OSPF-Hello Pakete erstellt hat. Als weiterer Input dient ein weiteres Text-File, welches Link State Updates aufführt, welche von jedem Router aus der Nachbarschaft geschickt werden. Mit diesen Informationen wird die Routing Tabelle berechnet, welche in ein File geschrieben wird.

### Hintergrund

Das Open Shortest Path First (OSPF) Protokoll ist ein Link-State-Routing-Protokoll für IP-Netzwerke und basiert auf den Dijkstra-Algorithmus. Das OSPF Protokoll sammelt Link-State Informationen von den verfügbaren Routern und erstellt so eine Topologie des Netzwerks. Das Routing Protokoll berechnet den kürzesten Pfad anhand der «Link-Kosten» welche zu jedem Knoten im Netzwerk gespeichert wurden.

## Wichtige Begriffe<sup>1</sup>

### Hello Paket / Hello Protokoll:

Das Hello-Protokoll ist in OSPF First für den Netzbetrieb ein integraler Bestandteil des gesamten Routingprozesses. Es ist verantwortlich für:

- Senden von Keepalives in bestimmten Intervallen (damit wird bestätigt, ob die Route noch besteht)
- Zur Entdeckung eines Nachbarn
- Aushandlung der Parameter
- Wahl eines Designated Routers (DR) und des Backup-DRs

### Link State Updates / Link State Advertisements:

OSPF-Router tauschen Informationen über die erreichbaren Netze mit sogenannten LSA-Nachrichten (Link State Advertisements) aus. Es gibt insgesamt über 10 verschiedene LSA Typen. In diesem Projekt konzentrieren wir uns aber nur auf den folgenden:

- Router-LSA (Typ 1): Für jeden aktiven Link des Routers, wird ein Eintrag im Router-LSA erzeugt. In ihm wird neben der IP-Adresse des Links auch die Netzmaske des Links und der Netzwerktyp (Loopback, Point-to-Point, normales Netz) eingetragen.

Ein Link State Update kann mehrere solche Link State Advertisements beinhalten

## Erstellung Nachbarschaftstabelle

Ein Router der OSPF ausführt erstellt mit Hilfe des Hello Protokolles eine Nachbarschaftstabelle. Diese könnte folgendermassen aussehen:

Address	Interface	State	ID	Priority	Dead
192.168.1.2	so-0/0/0.0	Full	R2	128	36
192.168.1.3	so-0/0/1.0	Full	R3	128	38
192.168.1.4	so-0/0/2.0	Full	R4	128	33

Um das ganze ein wenig zu vereinfachen, werden wir diesen Schritt weglassen und gehen davon aus, dass die Nachbarschaften bereits aufgebaut wurden. Unser Programm (welches einen Router repräsentiert) liest eine solche Nachbarschaftstabelle aus einer Textdatei ein.

## Erstellung Topology Table

Nachdem die Nachbarschaften ermittelt wurden, erstellt der Router anhand von Link State Updates eine Topology Table (oder auch Topology Graph / Link State Database). Dazu müssen erst mal die Link State Updates ausgewertet werden. In unserem Projekt wird ein solches Update als Datei im Filesystem repräsentiert. Jede solche Datei repräsentiert wiederum ein Link State Update eines benachbarten Routers.

Eine solche Datei beinhaltet eine Kette von Hexadezimalen Werten und weist folgende Struktur auf (eine Zeile sind 4 Byte):

---

<sup>1</sup> Quelle: [http://de.wikipedia.org/wiki/Open\\_Shortest\\_Path\\_First](http://de.wikipedia.org/wiki/Open_Shortest_Path_First)









0	7	8	15	16	23	24	31
---	---	---	----	----	----	----	----

Destination MAC															
Source MAC															
Type															

Version	Header Length	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source IP				
Destination IP				

OSPF Version (2)	Message Type (4 = LS Update Packet)	Packet Length
Router ID		
Area ID		
Packet Checksum	Auth Type	
Auth Data		

Number of LSAs		
LS Age	Options	LSA Type (1 = Router-LSA)
Link State ID		
Advertising Router		
LS Sequence Number		
LS Checksum	Length	
Flags	Empty	Number of Links
IP Network / Subnet Nr		
Link Data		
Link Type	Number of TOS metrics	TOS 0 Metric
More Links...		
More Link State Advertisements...		

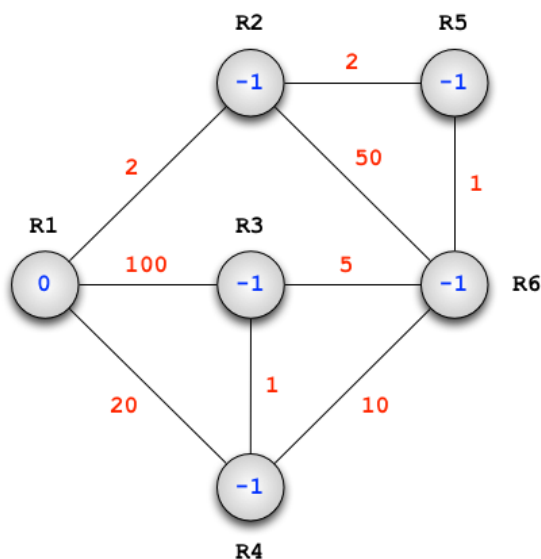
	Bits		Data Link Frame Header		IP Packet Header		OSPF Packet Header
	Link State Update Packet		Link State Advertisement Header		Link State Advertisement Content		Links

Nachdem die Link State Updates eingelesen und ausgewertet wurden, wird daraus die Topologie Tabelle erstellt. Diese stellt dar, welche Router miteinander verbunden sind und wie hoch die Kosten dieser Verbindungen sind.

Als Tabelle:

From Router (Router ID)	To Router (Router ID)					
	R1	R2	R3	R4	R5	R6
R1		2	100	20		
R2	2				2	50
R3	100			1		5
R4	20		1			10
R5		2				1
R6		50	5	10	1	

Als Graph:



In Haskell wird diese Tabelle bzw. dieser Graph als Liste mit folgendem Typ definiert:

```
type Graph = [(RouterId, [(RouterId, Distance)])]
```

Das erste Element im Tupel ist die Router ID, das zweite ist eine Liste von Tupel. Jedes dieser Tupel repräsentiert wiederum einen Nachbar Router. Im „Nachbar Tupel“ ist das erste Element die Router ID und das zweite die Metrik bzw. die Distanz die für den Dijkstra Algorithmus relevant ist.

Die Liste für das oben gezeigte Beispiel würde in Haskell also folgendermassen aussehen:

```
graphInput = [("R1", [("R2", 2), ("R3", 100), ("R4", 20)]),
              ("R2", [("R1", 2), ("R5", 2), ("R6", 50)]),
              ("R3", [("R1", 100), ("R4", 1), ("R6", 5)]),
              ("R4", [("R1", 20), ("R3", 1), ("R6", 10)]),
              ("R5", [("R2", 2), ("R6", 1)]),
              ("R6", [("R2", 50), ("R3", 5), ("R4", 10), ("R5", 1)])]
```

## Erstellung Routing Table

Der Router erstellt anhand der Topology Table eine Routing Table. Er berechnet jeweils den kürzesten Pfad zu einer Adresse basierend auf der konfigurierten Metrik, im Falle von OSPF ist dies die Bandbreite einer Verbindung. Um den kürzesten Pfad zu berechnen, erstellen wir aus den Daten einen gewichteten Graphen, wobei das Gewicht einer Kante aus der «cost metric», der Bandbreite einer Verbindung, bestimmt wird. In diesem Graphen berechnen wir für jeden Knoten den kürzesten Pfad, mithilfe des Dijkstra-Algorithmus.

## Metric

Laut Wikipedia definiert Cisco die Metrik wie folgt:  $10^8/\text{Bandbreite}$ . Damit wir in Haskell mit Integern arbeiten können definieren wir die Metrik leicht anders:  $10^{10}/\text{Bandbreite}$ .

Bandbreite	Cost Metric
10 Mbit/s	1000
100 Mbit/s	100
1 Gbit/s (Gigabit-Ethernet)	10
10 Gbit/s	1

## Aufbau Routingtabelle

Gateway of last resort is not set

```
141.108.0.0/16 is variably subnetted, 8 subnets, 3 masks
O 141.108.1.128/25 [110/65] via 141.108.10.10, 00:15:28, Serial0/0
O 141.108.9.128/25 [110/129] via 141.108.10.10, 00:15:28, Serial0/0
O 141.108.1.0/25 [110/65] via 141.108.10.10, 00:15:28, Serial0/0
C 141.108.10.8/30 is directly connected, Serial0/0
O 141.108.9.0/25 [110/129] via 141.108.10.10, 00:15:28, Serial0/0
O IA 141.108.10.0/30 [110/192] via 141.108.10.10, 00:15:28, Serial0/0
O 141.108.12.0/24 [110/129] via 141.108.10.10, 00:15:28, Serial0/0
O 141.108.10.4/30 [110/128] via 141.108.10.10, 00:15:29, Serial0/0
131.108.0.0/16 is variably subnetted, 9 subnets, 4 masks
C 131.108.4.128/25 is directly connected, Loopback1
O 131.108.5.32/27 [110/1010] via 131.108.1.2, 00:16:04, Ethernet0/0
O 131.108.33.0/24 [110/74] via 141.108.10.10, 00:15:29, Serial0/0
O 131.108.6.1/32 [110/11] via 131.108.1.2, 00:16:04, Ethernet0/0
C 131.108.5.0/27 is directly connected, Loopback2
O 131.108.6.2/32 [110/11] via 131.108.1.2, 00:16:06, Ethernet0/0
C 131.108.4.0/25 is directly connected, Loopback0
C 131.108.1.0/24 is directly connected, Ethernet0/0
O 131.108.26.0/24 [110/138] via 141.108.10.10, 00:15:31, Serial0/0
```

Listing 1 Beispiel einer Routing Tabelle. Quelle:

<http://www.ciscopress.com/articles/article.asp?p=26919&seqNum=7>

## Eintrag in einer Routing Table

O 141.108.1.128/25 [110/65] via 141.108.10.10, 00:15:28, Serial0/0

O	steht für das Routing-Protokoll, in unserem Fall immer O = OSPF
141.108.1.128/25	Ziel-Adresse
[110/65]	[Administrative Distanz/Metrik] Die Administrative Distanz des Protokolls ist bei OSPF 110 Die Metrik wird anhand der link cost berechnet und gibt das Gewicht eines Pfades an
via 141.108.10.10	Gibt an, ob die Route direkt oder über einen anderen Router erreichbar ist.
00:15:28	Dead-Time
Serial0/0	Interface Type

## Einschränkungen in unserem Projekt:

- Die Routing Table berücksichtigt nur OSPF-Routen
- Die Prefix-Länge einer Adresse wird nicht ausgewertet
- Wir beschränken uns auf Router in einer Area ohne Verbindungen in eine andere
- Wir verzichten auf die Bestimmung eines Designated und Backup Designated Routers
- Bei den Link State Advertisement Typen (LSA) beschränken wir uns auf LSA1 (Router-LSA)

## Source Code

Der Source Code ist direkt in den Source Files dokumentiert. Jede Funktion hat ihre eigene Beschreibung. Bei solchen mit vielen Parametern werden diese einzeln erläutert. Komplexere Funktionen haben ausserdem viele inline Kommentare die dabei helfen sollen den Überblick nicht zu verlieren und alles möglichst leicht nachvollziehbar zu machen.

In der Digitalen Version sind zudem noch die HTML Seiten des Haddock (äquivalent zu Java Dock) beigelegt, damit die Beschreibungen nicht im Code gesucht werden müssen.

## Dijkstra Algorithmus

Weil die Wahrscheinlichkeit sehr hoch ist, dass schon viele diesen Algorithmus in Haskell implementiert haben, mussten wir aufpassen, nicht aus versehen im Netz darüber zu stolpern. Also haben wir lediglich eine formale Beschreibung gesucht, und daraus einen auf unser Problem passenden Pseudocode erstellt:

1. Suche "Route" mit geringster Metrik in "Routes" dessen ID noch in "Graph" ist und entferne "Node" mit dieser ID aus "Graph"
2. Besuche alle "ChildNodes" in aufsteigender Reihenfolge
  - a. Für jedes "ChildNode": prüfe ob Metrik "ChildNode" > (Metrik "CurrentRoute" + Distanz "ChildNode")  
falls ja:
    - i. Aktualisiere Metrik "ChildRoute"
    - ii. Aktualisiere "RouteList" von "ChildRoute":  
"RouteList" von "CurrentNode" ++ [ID von "CurrentNode"]
3. Wiederhole bis "Graph" leer ist

An dieser Stelle sollte vielleicht noch erwähnt werden, dass wir zwischen „Distanz“ und „Metrik“ folgende Unterscheidung machen:

- Distanz: Die Kosten / Distanz zwischen zwei Knoten (direkte Verbindung).
- Metrik: Die aufsummierten Kosten / Distanz eine Route (z.B. von A über B bis C).

## Quellen:

- Skript «CCNA2 Routing-Konzepte und -Protokolle», Peter Gysel, IMVS FHNW
- Wireshark Beispiel-Capture
- The Routing Table v1.12 von Aaron Balchunas
- [http://en.wikipedia.org/wiki/Open\\_Shortest\\_Path\\_First](http://en.wikipedia.org/wiki/Open_Shortest_Path_First)
- [http://de.wikipedia.org/wiki/Open\\_Shortest\\_Path\\_First](http://de.wikipedia.org/wiki/Open_Shortest_Path_First)
- [http://en.wikipedia.org/wiki/Link-state\\_advertisement](http://en.wikipedia.org/wiki/Link-state_advertisement)
- [http://de.wikipedia.org/wiki/Link\\_State\\_Advertisement](http://de.wikipedia.org/wiki/Link_State_Advertisement)
- <http://de.wikipedia.org/wiki/Dijkstra-Algorithmus>
- [http://www.cisco.com/en/US/tech/tk365/technologies\\_white\\_paper09186a0080094e9e.shtml](http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a0080094e9e.shtml)
- <http://www.ciscopress.com/articles/article.asp?p=26919&seqNum=7>
- [http://www.routeralley.com/ra/docs/routing\\_table.pdf](http://www.routeralley.com/ra/docs/routing_table.pdf)
- <http://www.routeralley.com/ra/docs/ospf.pdf>
- [http://www.juniper.net/techpubs/en\\_US/junos9.6/information-products/topic-collections/nog-baseline/ospf-neighbors-introduction.html](http://www.juniper.net/techpubs/en_US/junos9.6/information-products/topic-collections/nog-baseline/ospf-neighbors-introduction.html)
- [http://www.cisco.com/en/US/tech/tk365/technologies\\_white\\_paper09186a0080094e9e.shtml](http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a0080094e9e.shtml)
- [http://www.tcpipguide.com/free/t\\_OSPFBasicTopologyandtheLinkStateDatabase.htm](http://www.tcpipguide.com/free/t_OSPFBasicTopologyandtheLinkStateDatabase.htm)
- <http://www.haskell.org/tutorial/modules.html>
- <http://www.haskell.org/cabal/users-guide/>
- <http://stackoverflow.com/questions/5804995/cabal-to-setup-a-new-haskell-project>
- <http://hackage.haskell.org/packages/archive/base/latest/doc/html/>