TT1 Praktikum 1 & 2 : Ausarbeitung

Carsten Noetzel, Armin Steudte

09.11.2011

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Projektschritt 1 - Aufbau

Im ersten Praktikumstermin wurden die Router konfiguriert und mit den nötigen Protokollen versehen. Die Versuchsdurchführung im zweiten Praktikumstermin wurde in Zusammenarbeit mit André Harms und Oliver Steenbruck durchgeführt und die Ergebnisse dokumentiert.
Der unter Abbildung 1 dargestellte Versuchsaufbau liegt den Versuchen zu Grunde. Hierbei wurde im Wechsel ein Router der Gruppe Noetzel, Steudte und der Gruppe Harms, Steenbruck aufgestellt und die Messungen zwischen den beiden äußersten Routern durchgeführt. Die Anzahl der Hops wurde mittels Tracert-Befehlen und bei OLSR über die Weboberfläche ermittelt. Dabei wurde angestrebt, dass die beiden äußersten Router über die beiden Router in der Mitte kommunizieren, was sich in den Versuchen durch geschickte Positionierung der Router auch als erfüllbar erwies.

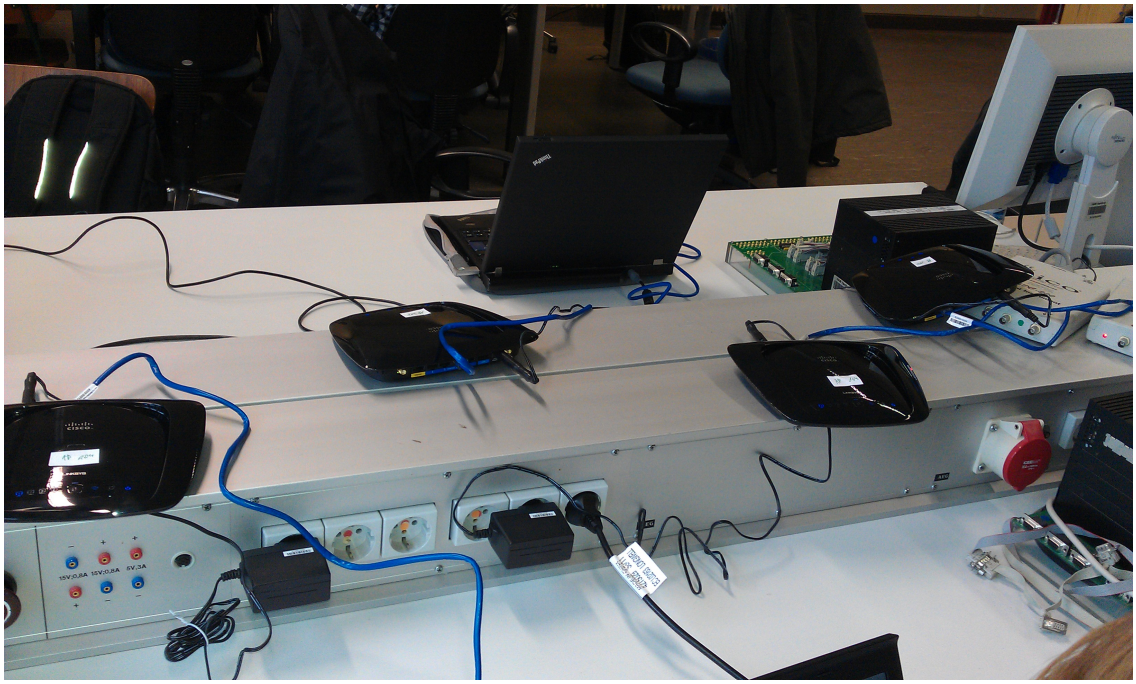

Abbildung 1: Versuchsaufbau mit 4 Routern

2 Projektschritt 2 - Vergleich Babel und OLSR

2.1 Babel

2.1.1 Informationsaustausch

Jeder Router A hält beim Babel-Protokoll die Kosten zu einem Nachbarrouter B in der Form C(A, B) vor. Die Metrik einer Route bestimmt sich über die Summierung aller Kosten zwischen den Knoten die auf der Route liegen. Das Ziel des Algorithmus liegt darin für jede Quelle S einen Baum der Routen, mit den niedrigsten Metriken zu S zu berechnen.

Babel nutzt *Hello-Messages* zur Durchführung des *Neighbour-Discovery-Processes*. Hello-Messages werden periodisch über alle Interfaces des Knoten gesendet um so die Nachbarknoten zu entdecken.
Zusätzlich werden Hello-Messages in Verbindung mit den *IHY-Messages* (I Heard You Messages)

genutzt, um die *Bidirectional Reachability* und die Metrik zwischen Sender- und Empfängerknoten zu ermitteln (vgl. Abbildung 2). Der Empfänger antwortet auf Hello-Messages mit den IHY-Messages. Diese erhalten die, mit Hilfe der Hello-Messages ermittelten, Laufzeitzeitverzögerungen aus Sicht des Senders der Hello-Messages. Mit Hilfe der Laufzeitverzögerungen findet dann die Bewertung der Qualität der Links, sowohl in Sende- als auch Empfangsrichtung statt.

Was aus Abbildung 2 hervorgeht, und uns so vorher nicht bewusst war, ist dass die Knoten zum Austausch von Nachrichten unter Babel *IPv6* nutzen. Die ausgetauschten Nachrichten enthalten dabei sowohl auf IPv6- als auch IPv4-Adressen und werden an eine IPv6-Multicast-Group gesendet.

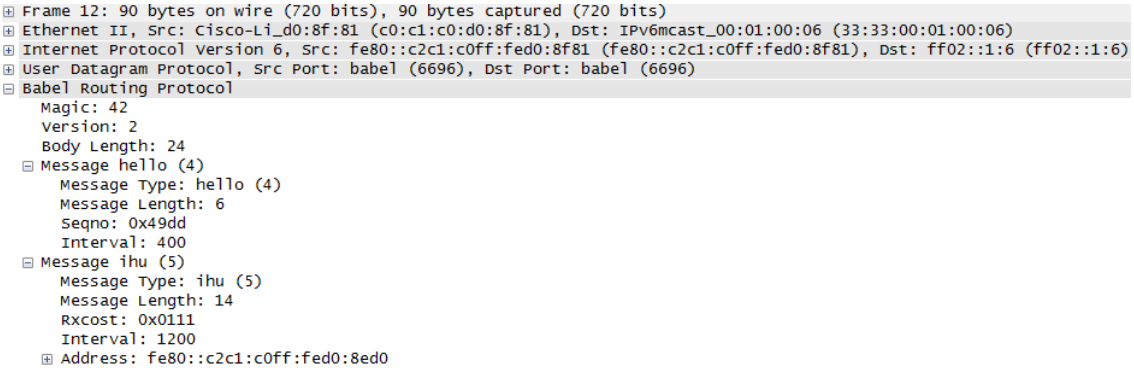

Abbildung 2: Beispiel für eine Babel-Hello-Message

2.1.2 Mesh - Konfiguration

Die lokalen Mesh-Konfigurationen werden über einen verteilten Bellman-Ford Algorithmus berechnetet.
Hierzu hält jeder Router A zwei Werte vor, zum einen die geschätzte Distanz zu S (bezeichnet als D(A)) und den Next-Hop-Router zu S (bezeichnet als NH(A)). Zu Beginn des Informationsaustausches ist D(A)=unendlich und NH(A) ist nicht definiert.
Periodisch sendet jeder Knoten B ein Update seiner Routen an alle seine Nachbarn mit dem Inhalt D(B), welcher die Distanz zu S angibt. Erhält ein Nachbar A von B ein Update, so prüft A ob B für S als Next-Hop-Router eingetragen ist. Ist dies der Fall wird als Distanz D(A) die Summe aus C(A,B)(= Kosten von A nach B) und D(B)(= Kosten von B zu S) gesetzt. Damit ist im Knoten A die Metrik von A nach S über B aktualisiert.
Im Fall, dass B nicht als Next-Hop-Router für S auf A eingetragen ist, vergleicht A die Summe aus C(A,B)(= Kosten von A nach B) und D(B)(= Kosten von B zu S) mit dem gegenwärtigen Wert von D(A). Ist C(A,B)+D(B) kleiner als D(A) ist die angebotene Route besser als die bisher eingetragene und NH(A) wird auf B und D(A) auf C(A,B)+D(B) gesetzt.
Im Pseudocode sieht dies in etwa so aus:

```
1 receiveRouteupdate(D(B)) from B for S
2 If NH(A) = B Then
3         D(A) = C(A,B)+D(B)
4 Else
5         If C(A,B)+D(B) < D(A) Then
6                 NH(A) = B
7                 D(A) = C(A,B)+D(B)
8         EndIf
9 EndIf
```

Durch den Austausch der Updates wird Mesh-Konfiguration festgelegt und die Knoten wissen an welchen Knoten sie Pakete weiterleiten müssen, um ein bestimmtes Ziel mit möglichst geringen geschätzten Gesamtkosten zu erreichen.

2.1.3 Loop-Verhinderungsstrategie

Da es beim Bellman-Ford Algorithmus nach einer Topologieänderung und den damit verbundenen Updates zum *Count-to-infinity*-Problem kommen kann, benötigt Babel eine Loop-Verhinderungs-Strategie.
Hierzu werden in Babel die sogenannten *Feasibility Conditions* eingeführt. Dabei werden, erhaltene Routenaktualisierungen verworfen, wenn diese nicht beweisen können, dass die Annahme des Updates zu keinem Loop führen. Router A hält dazu eine *Feasibility Distance* (bezeichnet als FD(A)) vor, welche als Wert die kleinste Distanz zu S enthält, die A jemals angeboten wurde. Ein Update von einem Nachbarn B ist zulässig (feasible), wenn die Metrik D(B) (= Kosten von B zu S) kleiner als As FD(A) ist.
Das diese Bedingung weniger restriktiv als die *DSDV-Feasibility* ist, macht das Beispiel in Abbildung 3 deutlich. A kommt über B zu S, womit D(A)=FD(A)=2 ist. Da D(C)=1 und damit kleiner als FD(A) ist, ist eine alternative Route über C für A zulässig, auch wenn die Metrik C(A,C)+D(C) mit 5 schlechter als die aktuelle Route ist. Für den Fall das die Route über B abbricht kann so die Alternativroute genutzt werden.
Die *DSDV-Feasibility* besagt, dass ein Update nur dann angenommen werden darf, wenn C(A,C)+D(C) kleiner oder gleich D(A) ist, wodurch die Alternativroute über C nicht zulässig wäre, da die DSDV-Bedingung nicht erfüllt ist (4+2 <= 2).
Um zu zeigen, dass die *Feasibility Condition* trotzdem noch Loop-Freiheit garantiert, sei zu beachten, dass wenn A ein Update von B akzeptiert D(B) nicht kleiner als FD(B) sein kann. Zudem gilt FD(B) ist kleiner als FD(A) da das Update für A zulässig ist und dieser es annimmt. Da diese Eigenschaft weiterhin erhalten bleibt, wenn A Updates versendet, ist sichergestellt, dass die Route keine Loops enthält.

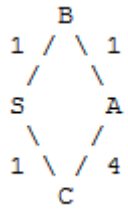

Abbildung 3: Beispiel Feasabiliy Condition aus RFC6126

Um zu verhindern, dass ein Router aufgrund fehlender zulässiger Routen ausgeschlossen wird, werden in Babel die Routen durchnummeriert. Ein Update von B an A ist damit zulässig, wenn entweder die *Feasibility Condition* D(B)<FD(A) erfüllt ist und die das Update die gleiche Sequenznummer besitzt oder aber wenn die Sequenznummer des Updates höher ist.

2.2 OLSR

Bei *OLSR* handelt es sich im Gegensatz zu Babel, wie der Name schon sagt, um einen Vertreter aus der Familie der Link-State-Routing Protokolle. Dabei enthält OLSR gegenüber anderen Vertretern die Optimierung, dass die Knoten ihre Informationen nur an ihre *Multipoint Relays (MPR)* senden müssen und diese das Verbreiten der Link-State Informationen übernehmen. Somit ermöglicht

OLSR ein effizientes *Flooding* der Informationen.
OLSR ist proaktiv, wodurch Routen unmittelbar vorhanden sind wenn diese gebraucht werden.

2.2.1 Informationsaustausch

Wie auch in Babel tauschen Nachbarknoten bei OLSR *Hello-Messages* aus. Die Nachrichten haben dabei drei Aufgaben:

- Erkennen von Links auf den Interfaces
- Neighbor Detection
- Bekanntgabe von MPRs

Ein Beispiel für eine Hello-Message ist in Abbildung 4 zu sehen. Zusätzlich werden auch noch

```
⊞ Frame 47: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
⊞ Ethernet II, Src: Cisco-Li_38:25:b4 (58:6d:8f:38:25:b4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol Version 4, Src: 10.0.0.115 (10.0.0.115), Dst: 10.0.0.255 (10.0.0.255)
⊞ User Datagram Protocol, Src Port: olsr (698), Dst Port: olsr (698)
⊟ Optimized Link State Routing Protocol
    Packet Length: 20
    Packet Sequence Number: 58924
  ⊟ Message: HELLO (LQ, olsr.org) (201)
      Message Type: HELLO (LQ, olsr.org) (201)
      Validity Time: 88,000 (in seconds)
      Message: 16
      Originator Address: 10.0.0.115 (10.0.0.115)
      TTL: 1
      Hop Count: 0
      Message Sequence Number: 6472
      Hello Emission Interval: 5,000 (in seconds)
      Willingness to forward messages: Unknown (4)
```

Abbildung 4: Beispiel für eine OLSR-Hello-Message

TC Messages, sogenannte *Topology Control Messages*, über alle OLSR-Interface gesendet (vgl. Abbildung 5). Sie enthalten Informationen über die lokale Topologie der Knoten und dienen zur Ermittlung der globalen Sicht auf das Netzwerk und dem Aufbau des Routing. Sie werden, wie bereits erwähnt, nur durch die MPRs weitergeleitet.
Sollte OLSR auf mehreren Interfaces gleichzeitig laufen werden auch noch *MID Messages (Main Adress and Multiple Interfaces)* gesendet, als zusätzliche Information zum Aufbau des Routing. Da in unserem Fall die Router jeweils nur über die WLAN-Interfaces OLSR ausführten, wurden keine MID Messages generiert.

Bei der Bewertung der Qualität von Links kann auf zwei Methoden zurückgegriffen werden. Standardmäßig wird eine gewisse Anzahl an erfolgreich gesendeten Hello-Messages als Grundlage für die Bewertung eines Links genutzt ob dieser als symmetrisch oder asymmetrisch eingestuft wird. Alternativ wird das *Link Hysteresis*-verfahren genutzt, wobei mit Schwellwerten gearbeitet wird, ab wann ein Link als stabil und damit als symmetrisch gilt. Beide Verfahren können Informationen aus dem Link-Layer für eine bessere Bestimmung mit einbeziehen. Außerdem existiert noch eine *Link Quality Extension*, die die Hello-Messages nutzt um eine Wahrscheinlichkeit für den Paketverlust auf einem Link zu berechnen. Pakete werde dann in diesem Fall über den Link mit der geringsten Wahrscheinlichkeit des Paketverlustes gesendet.
Wie in Abbildung 5 zu sehen ist, wird zu jedem Nachbarknoten ein Wert für die Link-Qualität in Sende- und Empfangsrichtung in der TC-Message mit gesendet. Diese heißen *LQ* und *NLQ* und weißen darauf in, dass in dem von uns eingesetzten *OLSR-Daemon* die Link Quality Extension aktiv war.

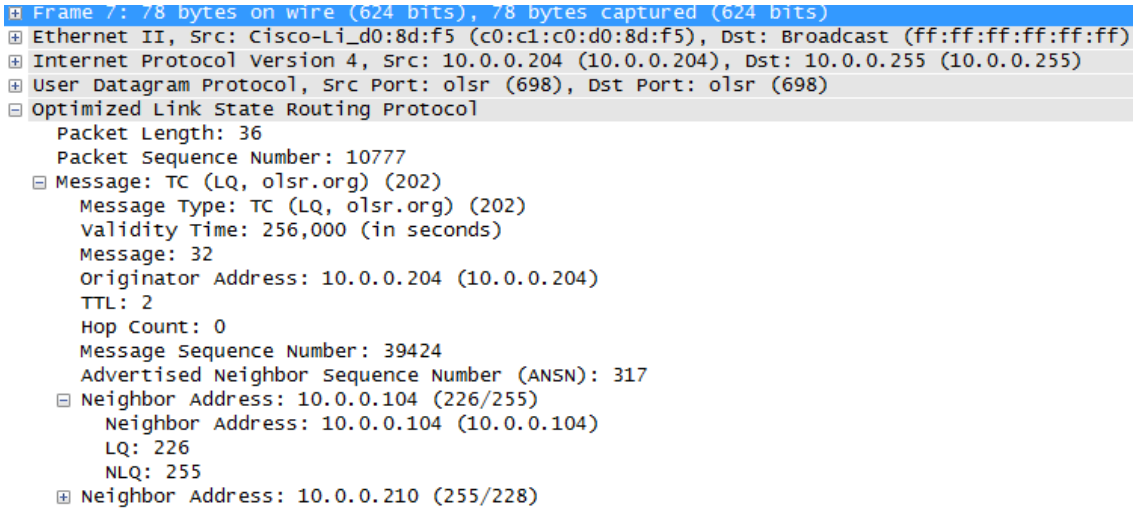

Abbildung 5: Beispiel für eine OLSR-TC-Message

2.2.2 Mesh - Konfiguration

Jeder Knoten im Netzwerk hält eine Routingtabelle mit folgenden Daten vor:

1. R-dest-addr R-next-addr R-dist R-iface-addr
2. R-dest-addr R-next-addr R-dist R-iface-addr
3. ,, ,, ,, ,,

Jeder Eintrag der Liste gibt an, welches Ziel (R-dest-addr) über welchen Next-Hop-Router (R-next-addr) erreichbar ist und welches lokale Interface (R-iface-addr) dafür genutzt werden muss. Ferner wird die Distanz vom lokalen Router bis zum Ziel (R-dist) in der Tabelle vorgehalten.
Die Daten basieren auf Informationen der *local link information base* und der Topologie, wodurch die Routingtabelle jedes Mal neu berechnet werden muss, wenn sich diese Daten ändern. Für jedes Ziel im Netzwerk zu dem eine Route bekannt ist, wird ein Eintrag in der Tabelle vorgehalten und bei Änderungen aktualisiert. Bei Aktualisierung wird die kürzeste Route zu einem Zielen mittels des Dijkstra-Algorithmus, auf Basis der aktuellen Sicht auf die Topologie, berechnet.

3 Projektschritt 3 - Vergleich der Übertragungsqualität

3.1 Versuchsaufbau

Der Versuchsaufbau ist unter Abbildung 1 auf Seite 2 zu sehen.
Die Router sind in folgender Reihenfolge aufgestellt (von links nach rechts): *192.168.204.1*, *192.168.210.1*, *192.168.104.1* und *192.168.110.1*. Die Messungen wurden über die äußerten Router durchgeführt, sodass sich mehrere Hops über die Router ergeben. Ein Tracert-Befehl von einem Client der mit dem Router *192.168.104.1* verbunden ist, an einen Client der mit dem Router *192.168.110.1* verbunden ist, ergibt die Route unter Abbildung 6.

Es wurde vor jedem Versuch sichergestellt, dass die Hop-Anzahl identisch ist, um vergleichbare Ergebnisse zu erzielen. Die Sendeleistung (TX-Power) wurde auf allen Geräten auf 1 gestellt, um möglichst viele Hops innerhalb unseres Versuchsaufbaus zu haben und nicht von anderen Gruppen gestört zu werden.

```
C:\Users\Carsten>tracert 192.168.110.2

Routenverfolgung zu 192.168.110.2 über maximal 30 Abschnitte

  1    <1 ms    <1 ms    <1 ms  192.168.204.1
  2     1 ms     1 ms     1 ms  10.0.0.210
  3     3 ms    13 ms     2 ms  10.0.0.104
  4     8 ms     3 ms     4 ms  10.0.0.110
  5     3 ms     4 ms     3 ms  192.168.110.2

Ablaufverfolgung beendet.
```

Abbildung 6: Route zwischen den Clients *192.168.104.2* und *192.168.110.2*

3.2 Babel

In diesem Abschnitt werden die Versuche vorgestellt, die mit dem Babel-Protokoll durchgeführt wurden.

3.2.1 Versuch 1 - Übertragungsqualität

Im ersten Versuch wurde die Übertragungsqualität von Babel gemessen, dabei wurden mit Hilfe des Programms *JPerf* 10 MB über das UDP-Transportprotokoll, von Client *192.168.110.2* an Client *192.168.104.2* übertragen.
Die Ausgaben von JPerf sind unter Abbildung 7 und Abbildung 8 zu sehen.
Es ist zu erkennen, dass die Bandbreite zwischen 612 kbits/s und 964 kbits/s schwankt und sich ein Jitter zwischen 11,841 ms und 24,471 ms ergibt. Der Paketverlust liegt zwischen 4,3 % und 95 %. Echtzeitanwendungen die einen Jitter kleiner als 50 ms fordern, werden damit zwar erfüllt, jedoch ist die Paketverlustrate mit durchschnittlich 80 % wesentlich höher als die geforderten 1 %.
Der Grund für die geringe Bandbreite, den hohen Jitter und die hohe Verlustrate, liegt in der geringen Sendeleistung (Tx-Power = 1) der Router. Eine Erhöhung der Sendeleistung, kann diese Werte verbessern (siehe Versuch 3).

```
bin/iperf.exe -s -u -P 0 -i 1 -p 5001 -f k
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
OpenSCManager failed - Zugriff verweigert (0x5)
[140] local 192.168.204.2 port 5001 connected with 10.0.0.110 port 44999
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[140]  0.0- 1.0 sec  99.1 KBytes   811 Kbits/sec  16.706 ms 1397567037/   71 (2e+009%)
[140]  1.0- 2.0 sec  94.7 KBytes   776 Kbits/sec  15.200 ms    3/   69 (4.3%)
[140]  2.0- 3.0 sec   118 KBytes   964 Kbits/sec  11.841 ms  129/  211 (61%)
[140]  3.0- 4.0 sec  86.1 KBytes   706 Kbits/sec  19.644 ms  491/  551 (89%)
[140]  4.0- 5.0 sec   106 KBytes   870 Kbits/sec  17.993 ms  354/  428 (83%)
[140]  5.0- 6.0 sec  90.4 KBytes   741 Kbits/sec  19.855 ms  938/ 1001 (94%)
[140]  6.0- 7.0 sec  79.0 KBytes   647 Kbits/sec  24.471 ms  881/  936 (94%)
[140]  7.0- 8.0 sec  87.6 KBytes   717 Kbits/sec  17.345 ms  401/  462 (87%)
[140]  8.0- 9.0 sec  89.0 KBytes   729 Kbits/sec  24.420 ms  930/  992 (94%)
[140]  9.0-10.0 sec  74.6 KBytes   612 Kbits/sec  24.101 ms  409/  461 (89%)
[140] 10.0-11.0 sec  81.8 KBytes   670 Kbits/sec  22.367 ms 1039/ 1096 (95%)
[140]  0.0-11.9 sec  1091 KBytes   752 Kbits/sec  136.438 ms 7745/ 8505 (91%)
```

Abbildung 7: JPerf Protokoll Babel

3.2.2 Versuch 2 - Failover

3.2.3 Versuch 3 - Erhöhung der Sendeleitung

In diesem Versuch wurde lediglich die Sendeleistung des Router *192.168.110.1* auf den Wert 11 gesetzt, um die Auswirkung einer erhöhten Sendeleistung zu untersuchen. Dabei wurde zunächst festgestellt, dass durch Erhöhung der Sendeleistung der Hop-Count zwischen den Clients um einen Hop abnimmt. Da der Router *192.168.110.1* mit erhöhter Sendeleistung sendet, kann er Router

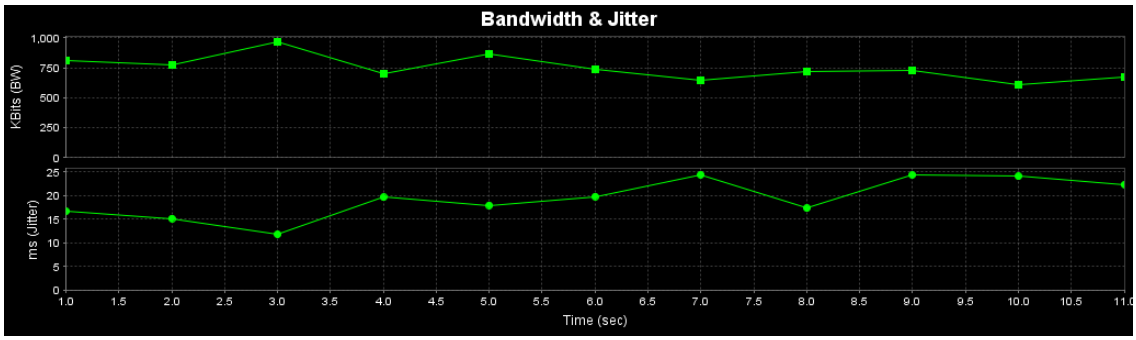

Abbildung 8: JPerf Bandbreiten und Jitter Graph - Babel

192.168.210.1 direkt erreichen und muss nicht mehr über Router *192.168.104.1* springen.
Es wurden wieder 10 MB mittels UDP von von Client *192.168.110.2* an Client *192.168.104.2* übertragen. Die Ergebnisse sind in Abbildung 9 und Abbildung 10 zu sehen.
Die Bandbreite liegt hierbei zwischen 4386 kbits/s und 5715 kbits/s und ist damit deutlich höher als im ersten Versuch. Auch beim Jitter, der in diesem Versuch zwischen 2,950 ms und 5,214 ms liegt, sind deutliche Verbesserungen festzustellen. Die Paketverlustrate liegt trotzdem noch zwischen 38 % und 58 %.
Trotz Erhöhung der Sendeleistung eines Routers, kann die Echtzeitanforderung mit einem maximalen Paketverlust von 1 % nicht erfüllt werden. Es bleibt zu überprüfen, ob eine Erhöhung der Sendeleistung der anderen Router eine Verbesserung im Bereich der Verlustrate erwirkt.

```
bin/iperf.exe -s -u -P 0 -i 1 -p 5001 -f k
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
OpenSCManager failed - Zugriff verweigert (0x5)
[140] local 192.168.204.2 port 5001 connected with 10.0.0.110 port 42737
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[140]  0.0- 1.0 sec   653 KBytes  5351 Kbits/sec  4.191 ms 1397567070/  490 (2.9e+008%)
[140]  1.0- 2.0 sec   590 KBytes  4833 Kbits/sec  3.724 ms  275/  686 (40%)
[140]  2.0- 3.0 sec   564 KBytes  4622 Kbits/sec  3.217 ms  462/  855 (54%)
[140]  3.0- 4.0 sec   590 KBytes  4833 Kbits/sec  3.743 ms  560/  971 (58%)
[140]  4.0- 5.0 sec   604 KBytes  4951 Kbits/sec  4.005 ms  315/  736 (43%)
[140]  5.0- 6.0 sec   535 KBytes  4386 Kbits/sec  3.398 ms  425/  798 (53%)
[140]  6.0- 7.0 sec   637 KBytes  5221 Kbits/sec  3.855 ms  543/  987 (55%)
[140]  7.0- 8.0 sec   698 KBytes  5715 Kbits/sec  3.417 ms  302/  788 (38%)
[140]  8.0- 9.0 sec   551 KBytes  4516 Kbits/sec  2.950 ms  431/  815 (53%)
[140]  9.0-10.0 sec   616 KBytes  5045 Kbits/sec  3.472 ms  420/  849 (49%)
[140]  0.0-10.3 sec  6311 KBytes  4998 Kbits/sec  5.214 ms 4101/ 8497 (48%)
```

Abbildung 9: JPerf Protokoll Babel mit erhöhter Sendeleistung

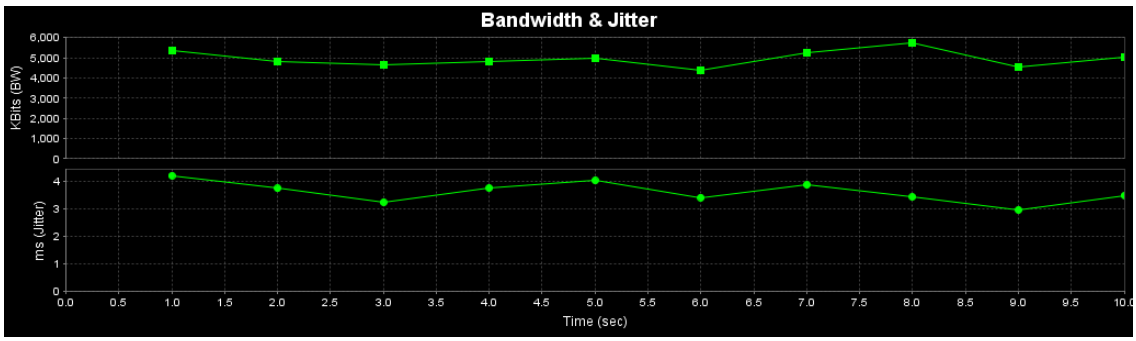

Abbildung 10: JPerf Bandbreiten und Jitter Graph - Babel mit erhöhter Sendeleistung

3.3 OLSR

Dieser Abschnitt befasst sich mit den Versuchen, die mit dem OLSR-Protokoll durchgeführt wurden.

3.3.1 Versuch 1 - Übertragungsqualität

Im ersten Versuch wurden mittels JPerf 10 MB Daten von Client *192.168.110.2* an Client *192.168.104.2* übertragen. Als Transportprotokoll wurde hierbei UDP verwendet und für alle Router die Sendeleistung (TX-Power) auf den Wert 1 gesetzt.
Die Brandbreite liegt bei OLSR zwischen 400 kbits/s und 2458 kbits/s, der Jitter zwischen 5,852 ms und 28,830 ms. Die Paketverlustrate liegt zwischen 11 % und 93 %.
Im Vergleich zum Babel-Protokoll weist das OLSR-Protokoll teilweise bessere Werte bezüglich der Bandbreite und des Jitters auf, aber die Werte schwanken stärker zwischen den minimalem und maximalen Messwerten.
Es liegt die Vermutung nahe, dass die Schwankungen am größeren Bandbreitenbedarf liegen, die für den Austausch der Topologie-Informationen benötigt werden. Da im Vergleich zu Babel jeder OLSR Router die komplette Topologie vorhält, ist der Informationsaustausch deutlich erhöht. Nicht auszuschließen sind auch Fremdeinwirkungen und es ist zudem zu beachten, dass mit der geringen Sendeleitung der Router an Grenzen gearbeitet wird, an denen die Übertragung unzuverlässig ist.

```
bin/iperf.exe -s -u -P 0 -i 1 -p 5001 -f k
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
OpenSCManager failed - Zugriff verweigert (0x5)
[140] local 192.168.204.2 port 5001 connected with 10.0.0.110 port 51995
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[140]  0.0- 1.0 sec   123 KBytes  1011 Kbits/sec  10.517 ms 1397567041/   92 (1.5e+009%)
[140]  1.0- 2.0 sec  48.8 KBytes   400 Kbits/sec  28.830 ms    4/   38 (11%)
[140]  2.0- 3.0 sec  61.7 KBytes   506 Kbits/sec  23.604 ms    9/   52 (17%)
[140]  3.0- 4.0 sec   132 KBytes  1082 Kbits/sec  21.529 ms 1045/ 1137 (92%)
[140]  4.0- 5.0 sec   214 KBytes  1752 Kbits/sec  7.903 ms 1884/ 2033 (93%)
[140]  5.0- 6.0 sec   207 KBytes  1693 Kbits/sec  11.123 ms  680/  824 (83%)
[140]  6.0- 7.0 sec   215 KBytes  1764 Kbits/sec  12.469 ms  745/  895 (83%)
[140]  7.0- 8.0 sec   223 KBytes  1823 Kbits/sec  12.099 ms  711/  866 (82%)
[140]  8.0- 9.0 sec   237 KBytes  1940 Kbits/sec  8.921 ms  666/  831 (80%)
[140]  9.0-10.0 sec   151 KBytes  1235 Kbits/sec  11.799 ms  527/  632 (83%)
[140] 10.0-11.0 sec   300 KBytes  2458 Kbits/sec  5.852 ms  890/ 1099 (81%)
[140]  0.0-11.0 sec  1918 KBytes  1427 Kbits/sec  7.206 ms 7168/ 8498 (84%)
[140]  0.0-11.0 sec  1 datagrams received out-of-order
```

Abbildung 11: JPerf Protokoll OLSR

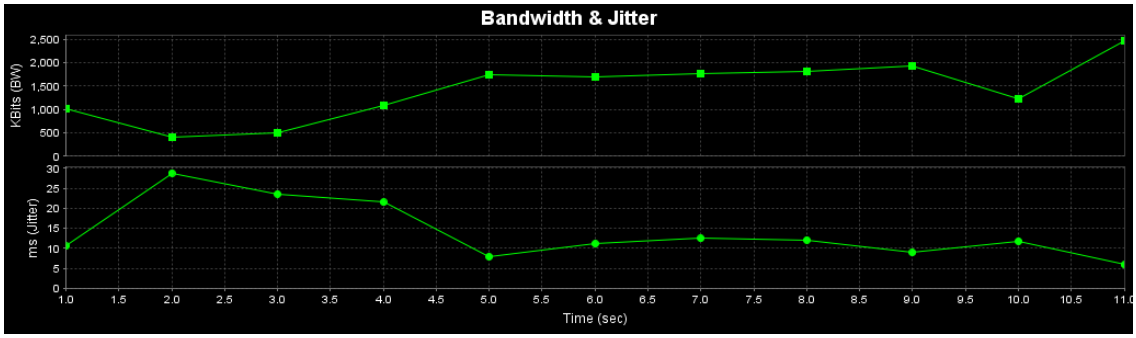

Abbildung 12: JPerf Bandbreiten und Jitter Graph - OLSR

3.3.2 Versuch 2 - Failover