ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Evaluation of Scheduling Methods over Multipath Routing in Wireless Mobile Ad Hoc Networks

Semester Thesis, SA-2006-26
SS 06, April 2006 - July 2006

Authors: Regula Gönner
Dominik Schatzmann

Professor: Bernhard Plattner
Advisor: Georgios Parissidis

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Summer Semester 2006

Semester Thesis

for

Regula Gönner (D-ITET), Dominik Schatzmann (D-ITET)

Main Reader: Georgios Parissidis
Alternate Reader: Martin May

Issue Date: 3. April 2006
Submission Date: 30. June 2006

Evaluation of scheduling methods over multipath routing in wireless mobile Ad Hoc networks

1 Introduction

Multi-path routing is not a new concept and has been proposed and implemented in packet and circuit switched networks. In circuit switched telephone networks alternate path routing was proposed in order to increase network utilization as well as to reduce the call blocking probability. However, the wide deployment of multi-path routing is so far prevented due to higher complexity and the additional cost of extra routes that need to be stored in the routers.

Analysis and comparison of single path and multi-path routing with load balance mechanisms in ad hoc networks in terms of protocol overhead, traffic distribution and throughput has been conducted in [1]. The results reveal that in comparison with single-path routing, multi-path routing achieves higher throughput and increases network capacity. As the dimensions of mobile ad hoc networks are spatially bounded network congestion is inherently experienced in the center of the network as shortest paths mostly include nodes that are located at the center of the network. Thus a protocol in order to route data packets over non-congested links and maximize overall network throughput should target at utilizing the maximum available capacity of the calculated multiple routes. The authors concluded that routing or transport protocols in ad hoc networks should provide appropriate mechanisms to push the traffic further from the center of the network to less congested links.

In [2] the effect of the number of multiple paths on routing performance has been studied using an analytical model. Their results showed that multi-path routing performs better than single path if the

number of alternative paths is limited to few paths. Simulation results of their model demonstrated that with multi-path routing end-to-end delay is higher as alternate paths tend to be longer.

The goal of the present thesis is to conduct a performance evaluation of scheduling algorithms over multipath routing for mobile wireless Ad Hoc networks. Therefore, the main objectives of the present thesis are:

1. Implement and conduct a performance evaluation of scheduling algorithms (Round Robin, Weighted Round Robin, Proportional selection of paths) over a multipath routing protocol in mobile wireless Ad Hoc networks.

2. Propose and implement an optimized scheduling algorithm tailored for mobile wireless Ad Hoc networks that maximizes the throughput as well as minimizes the average end-to-end delay.

2 Conceptual Formulation

Wireless ad hoc networks consist of many features that differentiate them from conventional wired networks. The non-employment of multi-path routing in the standardized routing protocols used in the Internet today does not imply that multi-path routing is not an appropriate and promising solution for wireless mobile ad hoc networks. The unreliability of the wireless medium, the unpredictability of the network topology as node failures may occur frequently and the dynamic topology due to mobility of nodes result to frequent path breaks, network partitioning and high delays of path re-establishments.

Multi-path routing represents a promising routing method for wireless mobile ad hoc networks. Multi-path protocols establish multiple disjoint paths to the same destination. Thus, the probability of disconnection from the sender to the receiver is effectively reduced. Multi-path routing achieves load balancing of data traffic across network elements which is especially important in resource constrained environments, like mobile wireless ad hoc networks, where power is scarce and limited. Furthermore, multi-path routing attains more efficient protection to attacks. Spreading traffic over multiple paths, attacks such as packet interception and traffic analysis are much harder to perform. A multi-path routing protocol should fulfill the following properties:

- The routing protocol must provide multiple, loop-free and preferably node-disjoint paths to destinations. These two properties represent a strong measure of path independence.

- The multiple paths should be used simultaneously for data transport.

- Multiple routes need to be known at the source.

A multi-path routing protocol for mobile ad hoc networks that satisfy the above-mentioned properties is the AOMDV [3]. AOMDV extends the AODV [4] to provide multiple paths. In AOMDV each RREQ and respectively RREP defines an alternative path to the source or destination. Multiple paths are maintained in routing entries in each node. The routing entries contain a list of next-hops along with corresponding hop counts for each destination To ensure loop-free paths AOMDV introduces the *advertised_hop_count* value at node *i* for destination *d*. This value represents the maximum hop-count for destination *d* available at node *i*. Consequently, alternate paths at node *i* for destination *d* are accepted only with lower hop-count than the *advertised_hop_count* value.

An implementation of the AOMDV protocol in *ns* is available, however in the original implementation only one path is used at a time. Alternative paths are used as backup paths. Therefore, a modification of the original AOMDV source code should be done to implement the studied scheduling methods. Primarily, for the first part of the thesis we consider the following scheduling algorithms:

1. **Round Robin:** In this scheduling algorithm, a path is selected with the same probability among the multiple paths at the time a data packet is sent. Therefore, assuming that at time t, n paths are known at a sender s towards the destination d. A path i is selected with probability p_i:

$$p_i = \frac{1}{n}, \sum_{i=1}^{n} p_i = 1, i \in [1, n]. \quad (1)$$

2. **Weighted Round Robin:** The weighted Round Robin algorithm represents an special case of the Round Robin algorithm. On each path i a weight w_i is assigned and accordingly a path is selected with probability p_i:

$$p_i = w_i, \sum_{i=1}^{n} w_i = 1, i \in [1, n]. \quad (2)$$

3. **Proportional selection of paths:** In this algorithm a proportion of available multiple paths is selected at each sender to disseminate data packets towards a destination. We assume two policies in the way paths are selected:

   a) N out of M. Select *best* N out of total M paths.

   b) Use only the best (N=1) out of M paths and keep the rest as backup.

The optimal selection of paths on the *"Proportional selection of paths"* algorithm, as well as the optimal assignment of weights for the *"Weighted Round Robin"* algorithm depend on parameters that inherently influence the performance of each algorithm:

- Path duration/lifetime. The total time a path is active.
- Path length. The total number of hops between a source and a destination.
- Path breakage probability. The probability that a path will be active at the time a packet is scheduled to be sent.

The second part of the thesis complements the performance evaluation of the studied scheduling algorithms. An optimized scheduling algorithm tailored for wireless mobile Ad Hoc that outperforms the abovementioned algorithms should be proposed and implemented.

2.1 Tasks

Summarizing, the essential tasks that need to be addressed in the present thesis are the following:

1. Performance evaluation of the abovementioned algorithms in the *ns* network simulator [5]. The performance evaluation should compare the different algorithms using the following metrics:
   a) Throughput: The ratio of the total number of packets delivered at the destination to the total number of data packets sent.
   b) Average end to end delay: The average end-to-end delay consists of propagation, queuing, retransmission at the MAC layer, and buffering delays for successfully delivered data packets.
2. Design and implementation of an optimized scheduling algorithm(s) over multiple paths for wireless mobile Ad Hoc networks that achieve higher throughput and lower end-to-end delay.

A detailed timeline and a detailed list including a short description of each task is presented in table 1. Tasks are *distributed evenly* in two categories: ***A*** and ***B***; one per student.

Tasks	*Time*	*Task Description*	*A*	*B*
1.	1 week (3 April)	Related work	*	*
2.	1 week (10 April)	Getting familiar with *ns*	*	*
3.	1 week (17 April)	Installation/of a multipath routing protocol in *ns*	*	*
4.	2 weeks (24 April)	Implementation of scheduling algorithms		
4.1	*24.04 - 01.05*	- Round Robin	*	
4.2	*24.04 - 01.05*	- Proportional selection of paths		*
4.3	*01.05 - 08.05*	- Weighted Round Robin (incl. parameters)	*	
4.4	*01.05 - 08.05*	- Proportional selection of paths (incl. parameters)		*
5.	2 weeks (8 May)	Performance evaluation of scheduling algorithms		
5.1	*08.05 - 22.05*	- RR + WRR	*	
5.2	*08.05 - 22.05*	- Prop. selection + Prop. selection (+ par.)		*
6.	2 weeks (22 May)	Proposal of optimized scheduling algorithms		
6.1	*22.05 - 05.06*	- Based on 4.1, 4.2	*	
6.2	*22.05 - 05.06*	- Based on 4.2, 4.3		*
6.3	*22.05 - 05.06*	- Intermediate Report	*	*
7.	1 week (05 June)	Performance evaluation of proposed scheduling algorithms	*	*
8.	1 week (12 June)	Assemble of the results	*	*
9.	2 weeks (19 June)	Thesis report	*	*
10.	30 June	Submission of final thesis report	*	*

Table 1: Timeline of the thesis

3 Important remarks

1. A final timeplan for the realization of the semester thesis should be made by the end of the first week and discussed with the supervisor.
2. By the end of the second month a short intermediate report should be composed and reviewed during a meeting of the students with the supervisors. The intermediate report should list the already achieved tasks and the tasks that are foreseen to have been accomplished by the end of the thesis. Strictly speaking the intermediate report should comply with a structural design of the final report (in a bulleted form).
3. An ordinary contact (at least once a week) between the students and their supervisor has to take place via telephone, e-mail, meetings, tikiwiki semester thesis web page or other means. During

these contacts the progress of the performed work has to be shown and problems should be discussed. Especially important is the daily reading of e-mails.

4 Thesis Results

A fifteen minutes presentation should be given in TIK Institute. The exact date of the presentation will be specified late in the summer semester. Apart from this presentation, the following documents should be handed in:

1. A detailed technical report (Bericht) in English. The following topics should be thoroughly addressed in this report: a description of the investigated research area, a description of the examined scheduling algorithms, a listing of the solved and unsolved problems (together with the reasons why they haven't been solved), references to literature, table of contents, figures,tables and potential appendices (glossary, programming code, etc.). The report should end up with an evaluation of how far the initial tasks of the thesis have been achieved and whether the initial timeplan was fulfilled. Five copies of the final report should be handed in, all bound and double sided printed. The technical report is preferred to be composed in Latex.
2. An abstract in both German and English, 1-2 pages long. This should contain a quick overview of the performed work. The structure of the abstract should be in the form: (1) Introduction, (2) Aims & Goals, (3) Results, (4) Future Work.
3. An electronic version of the technical report as well as of all the produced documents (code documentation, models etc.). Figures contained in the final report have to be additionally stored as independent data in a custom-selected format (ex. EPS). The material in electronic form should be either stored on a CD or in a separate directory on an institute's server (accounts should then be created for the students).
4. Referenced and processed literature, whether in electronic or printed form.
5. The complete source code of the system and of the test codes, together with all the necessary libraries/external programs.

References

[1] P. Pham and S. Perreau, "Performance analysis of reactive shortest path and multi-path routing mechanism with load balance," *INFOCOM, San Francisco, CA, USA*, 2003.

[2] A. Nasipuri, R. Castaneda, and S. R. Das, "Performance of multipath routing for on-demand protocols in mobile ad hoc networks," *Mob. Netw. Appl.*, vol. 6, no. 4, pp. 339–349, 2001.

[3] M. Marina and S. Das, "On-demand multipath distance vector routing in ad hoc networks," *Proceedings of the International Conference for Network Procotols (ICNP)*, November 2001.

[4] C. E. Perkins, E. M. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," *RFC 3561*, July 2003.

[5] "ns-2: Network simulator," http://www.isi.edu/nsnam/ns/.

Abstract

In a wireless mobile ad hoc network multiple routes towards a destination can be established. Multipath routing protocols aims at achieving a higher throughput in the network. One method is to use the additional paths as back-up which can reduce the routing overhead and increase the performance [1]. With an enhanced selection of the paths (scheduling) the throughput of the network can be further increased.
This thesis covers the design of a scheduling algorithm for the multipath routing protocol AOMDV. The first part of the work deals with the implementation of a general scheduling method e.g. round robin. The performance analysis of these schedulers, based on a scenario with random movement, did not show clear trends for further optimisation.
For this reason the second part of the work uses a simplified, static scenario that consists of 61 nodes in a hexagonal structure and 19 additional nodes placed in a small area (node density hotspot scenario). The traffic is assumed to be uniformly distributed. Node density is used as scheduling metric. Based on this specific scenario the following points has been investigated.

- Where is the bottleneck of the network and how is it related with the scheduling metric?
- How many paths are known from the sender towards the destination. Is scheduling possible and efficient?
- Which path can be found during the simulation? Do all paths go through the bottleneck?

The investigation with the help of simulations showed that the bottleneck is not at the place with the highest node density. The number of paths that are available on average converges for longer paths (more than 3 hops) to one. Due to this result, scheduling for long paths is not possible. The paths, that are found for different route requests, differ in space.
Due to the not sufficient number of paths for longer distances an alternative routing protocol should be used and the above mentioned questions should be answered for the new protocol.

Zusammenfassung

In einem wireless mobile ad hoc Netzwerk können verschiedene Pfade für eine Verbindung aufgebaut werden. Dies nutzen Multipath Routing Protokolle aus, um einen besseren Durchsatz des gesamten Netzwerkes zu erreichen. Eine Methode besteht darin die zusätzlichen Pfade als Back-up zu verwenden. Dadurch wird der Routing Overhead verkleinert und der Durchsatz gesteigert [1]. Durch ein verbessertes Auswahlverfahren der Pfade (Scheduling) kann der Durchsatz des Netzwerkes weiter optimiert werden.
Diese Arbeit setzt sich sich mit dem Entwurf eines Scheduling-Algorithmus für das Multipath Routing Protokoll AOMDV auseinander. Der erste Teil der Arbeit befasst sich mit der Implementierung eines allgemeinen Scheduling-Verfahrens, mit dessen Hilfe verschiedene Scheduler wie z.B. Round Robin simuliert werden können. Die Performanceanalyse dieser Scheduler basierend auf einem Random Movement Scenario liefert jedoch keinen klaren Trends für weitere Optimierung.
Aus diesem Grund wird im zweiten Teil der Arbeit ein vereinfachtes statisches Scenario verwendet. Dieses besteht aus 61 Knoten die in einer sechseckigen Struktur angeordnet sind. Weitere 19 Knoten werden innerhalb einer kleinen Fläche verteilt (Node Density Hotspot Scenario). Die Verbindungen werden als gleichverteilt angenommen. Als Scheduling Metrik wird Knotendichte (Node Density) verwendet. Anhand dieses spezifischen Szenarios werden folgende grundlegende Punkte untersucht:

- Wo befindet sich die Schwachstelle (Bottleneck) des Netzwerk und wie hängt diese mit der Scheduling Metrik zusammen?
- Wieviele Pfade kennt der Sender zum Ziel. Ist ein Scheduling möglich und effizient?
- Welche Pfade werden während der Simulation gefunden? Verlaufen alle Pfade durch die gleiche Schwachstelle?

Die Untersuchung mit Hilfe des Simulators zeigt, dass das Bottleneck nicht an der Stelle mit der höchsten Knotendichte auftritt. Die Anzahl Pfade, welche in der Quelle im Durchschnitt zur Verfügung stehen, konvergiert für länger Pfade (ab 3 Hops) gegen eins. Aus diesem Grund ist ein sinnvolles Scheduling für lange Pfade nicht möglich. Für verschiedene Route Requests werden meist solche Pfade gefunden, welche sich im Raum stark unterscheiden.
Aufgrund der ungenügenden Anzahl Pfade für längere Distanzen sollte ein anderes Routing Protokoll verwendet werden. Die oben erwähnten Punkte müssten für dieses Protokoll erneut getestet werden.

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

AODV	Ad hoc On demand Distance Vector
AODVM	Ad hoc On demand Distance Vector Multipath
AOMDV	Ad hoc On demand Multipath Distance Vector
ARP	Address Resolution Protocol
DSDV	Destination-Sequenced Distance-Vector
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
MSR	Multipath Source Routing
NFS	Network File System
RERR	Route Error
RR	Round Robin
RREP	Route Reply
RREQ	Route Request
RTT	Round Trip Time
WRR	Weighted Round Robin

1. Introduction

With the growing availability of wireless mobile devices, ad hoc networks are of increasing interest. Mobile ad hoc networks (MANET) consist of a collection of mobile wireless nodes and they transmit data without using fixed base stations or wired links. Typically the nodes have restricted processing power and memory resources. As the nodes have a limited communication range the path from source to destination has usually multiple hops and hence routing is crucial in MANETs.

As wireless mobile networks have different characteristics from wired networks therefore special routing protocols have to be deployed. MANETs are more dynamic than wired networks and they do not have dedicated nodes for control. To optimal use network resources, the routing protocol has to react fast on events like a node joining or leaving the network in order to fix invalid paths or to find new paths. Due to the limited bandwith of the MANET the routing overhead and the probability of packet collisions should be minimised.
In ad hoc networks often exist different paths from source to destination and the routing protocols can be divided into two different categories, those using only one path (singlepath routing) and those using multiple paths from source to destination (multipath protocols) [2].
Singlepath routing protocols can be divided into table-driven protocols and on demand routing protocols. The first establish and maintain routes to all available destinations while the latter set up a route if there is a demand for.

The table-driven routing protocols detect routes in advance and maintain this information even if a route is not used. An example of such a routing protocol is the Destination Sequenced Distance Vector (DSDV) routing protocol that periodically transmits its entire routing table to all the neighbours or sends an update if a topology change has been detected. The drawback of this protocol in mobile environments is that these broadcasts introduce routing overhead that increases with frequent topology changes.

Compared to table-driven protocols the routing overhead can be reduced if routes are only established if they are required (on demand protocols). An example of an on demand protocol is the Ad hoc On demand Distance Vector (AODV) routing protocol [3]. This protocol uses flooding to find a path from source to destination and in each hop is stored via which next hop a destination can be reached. Compared to table-driven protocols the waiting time for the connection setup is larger as the protocol first has to find a route to the destination.

Nodes in a MANET have limited transmission range and therefore a path from source to destination usually consists of multiple hops. As each link between those hops can break, the probability that a path breaks is higher the longer the path gets. The same holds for congestion and link quality. A singlepath routing protocol can reduce such effects only if it triggers new route searches, which increases the route overhead. Multipath routing protocols know multiple paths to the destination and can use them for load balancing or as a backup in case of a path break. Sending packets via different paths simultaneously can increase the reliability of the transmission as the effect of link failures is reduced but the receiver must be able to handle duplicate packets. The limitation of all the (multipath) protocols is given by the wireless medium. Even if multiple paths to a destination exist they can influence each other due to interference and will therefore not be totally independent.

It was shown that on demand multipath routing protocols can outperform table-driven singlepath protocols [4] and that the performance depends on the used scheduling algorithm. Different solutions based on source routing were presented in [5]. Source routing generates more routing overhead than on demand protocols and is less suited for highly mobile nodes.

This thesis uses Ad hoc On Demand Multipath Distance Vector an extension of the AODV protocol that

finds multiple paths to a destination and uses them as backup paths [1]. The AOMDV routing protocol has been extended with a set of scheduling functions to schedule over all paths and functions to evaluate the performance of the protocol. The performance of the scheduling methods is measured in terms of throughput and end-to-end delay and the evaluation is done using the network simulator *ns2*. In the following report we will give an introduction of MANETs and routing protocols and a survey on the related work. We will introduce the simulation environment and the scheduling methods together with all the scripts that have been developed to run simulations. Then all the evaluation tools and the performance evaluation will be presented. From these results conclusions will be drawn and different ideas for further work will be presented.

2. Theory

In the following part the concepts and backgrounds that are used for our work will be introduced. This consists of an introduction of mobile ad hoc networks, the routing protocol and the related work.

2.1. MANET

MANETs have four main issues to consider:

Predictability of environment MANETs may be deployed in unknown terrain under hazardous conditions or in hostile environments and as a consequence node failures may be frequent.

Unreliability of wireless medium Communication over a wireless medium is unreliable and error-prone and due to environmental conditions (interferences, weather) the quality of the link may be unpredictable. Resource restricted nodes may not be able to support transport protocols that ensure reliable communication.

Resource-constrained nodes Mobile nodes are often restricted in computing power, memory and battery power and the algorithms have to be energy-efficient and should operate with low processing power. The limited energy can reduce the available bandwidth as the node may not have enough power to operate at full link speed.

Dynamic topology As the nodes are mobile the topology of the network changes constantly.

As a result in ad hoc networks different types of errors can occur:

Transmission errors The unreliable wireless medium can destroy the transmitted packet [4].

Node failures Nodes may fail due to environmental conditions or they may drop out of the network voluntarily or due to lack of power.

Link failures Node failures, movement or changed environmental conditions may cause link breakages.

Route breakages Due to node or link failures the network topology changes and routes become outdated and thus incorrect and the routing protocol has to find a new route to the destination.

Congested nodes or links Due to the topology of the network and the nature of the routing protocol, certain nodes or links may become overutilised. This will lead to either larger delays or packet loss.

2.2. AODV

To route packets in a mobile ad hoc network a specialised routing protocol is required. For single routing a widely used protocol is the Ad hoc On demand Distance Vector routing protocol [3]. AODV is a reactive routing protocol, which means that routes are only computed if they are required.
If a source *S* wants to send data to destination *D* a route between these two nodes has to be found. For that purpose the source floods the network with RREQ packets (route request). Each node receiving the RREQ sends a route reply (RREP) if it knows a path to the destination (or it is the destination). If it does not know a path it rebroadcasts the request once. As soon as the source receives a RREP it can begin to transmit data. To determine the freshness of a route a destination sequence number is introduced, a larger sequence number indicates a newer route. All the nodes use received RREP and RREQ packets to update their routing tables if they contain newer information. If a node detects a broken link it sends a route error (RERR) packet as broadcast to all sources using this link.

2.3. AOMDV

If AODV is used and a path break occurs a new RREQ has to be sent. To reduce the routing overhead the multipath extension of AODV (AOMDV, Ad hoc On demand Multipath Distance Vector) [1] stores multiple paths to the destination and uses them as backup. As disjoint links break independently AOMDV establishes either link or node disjoint paths. To determine that a path is link disjoint each node forwarding a RREP has to check that the next and last hop of the path are unique. To get node disjoint paths the same check as with link disjoint paths is performed with the additional restriction that a node can only be part of one path.
To prevent routing loops node do not accept routes that are longer than the already advertised and they do not advertise shorter routes than the advertised. If a node receives a route advertisement with a higher sequence number than the one it already has, it discards all the routes and updates with the new information.

2.4. Related Work

As a larger transmission radius interferes with more nodes the effective available bandwidth per node is reduced if the transmission radius is enlarged. It was shown that for this trade-off the optimal number of neighbours is (rounded) six. A scenario with the nodes distributed in hexagonal form is therefore ideal, as the number of neighbours is six for all the nodes except those at the border. The problem of optimal node density in a mobile ad hoc network is covered in [6]. The increase of the transmission range of a node decreases the available effective bandwidth seen at individual nodes, but it increases the connectivity of the network, an effect that may be more important as node mobility increases. The results do not show a global optimum number of neighbours for all mobility scenarios, but as mobility increases the optimum shifts to higher connectivity and therefore the node density should increase as the rate of node movements increases.
[7] presents a load sensitive on demand routing approach that uses network load information to select paths. This routing method does not only include local load information but also the information about the load of all the neighbours. To obtain traffic information, the nodes continuously listen to neighbouring transmissions and hence the information can be gained without routing overhead.
[8] presents a protocol to balance the traffic load over the network. The proposed scheme is intended to route data packets circumventing congested paths. This reduces the end-to-end delay and balances the load over the network. To find the path with least traffic a cost function is used that takes the activity of neighbours as a measure for traffic interference and sums this along the path. To gather information about the neighbours *Hello* packets are used. The information about the traffic is forwarded with the route setup message and the destination can then select the path with the minimal cost and as a consequence the traffic is evenly distributed over the network.

3. Simulation Setup

In the performance evaluation the network simulator *ns2* version 2.26 has been used, as for this simulator already exists an implementation of the AOMDV protocol. In the following chapter the network simulator will be introduced, the scheduling functions will be described and the *Info* and *Ping* packets will be explained. The most important parameters to run the simulation are presented and a few important problems that have occurred are documented. To run the simulations more efficient *Condor* has been used and it will be explained below.
In Fig. 3.1 the structure of the simulation environment can be seen, the explanation of the different parts will be splitted into two chapters.

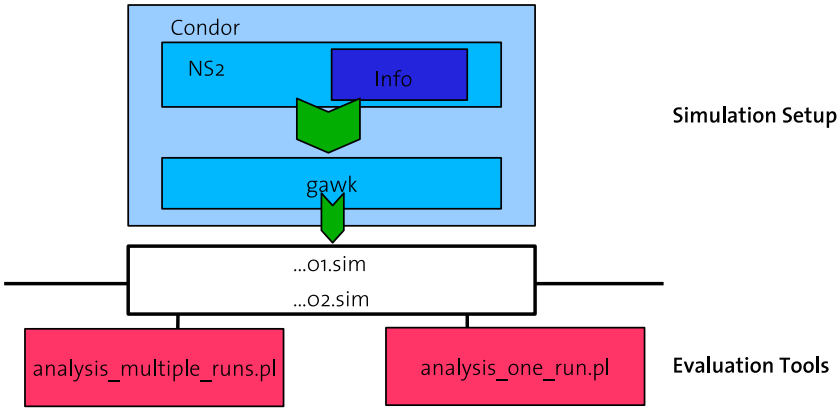

Figure 3.1: Structure of the simulation environment

3.1. Network Simulator

ns2 is a discrete event network simulator for network research and it is running on UNIX-like operating systems [9]. The simulator consists of two parts: an object oriented simulator in C++ and an OTcl part to execute command scripts of the user.
The C++ part is used for fast execution and therefore useful for running protocols. The OTcl part is used to configure the input via user scripts and it has the advantage that the entire system has not to be recompiled when an input has changed. In our simulation the Tcl script has been used to configure the simulation settings (such as nodes, traffic sources, channel type) and to start the simulation.

3.1.1. Problems

During the evolution of the thesis different problems have occurred. The two main problems concerning *ns2* are described below.

3.1.1.1. ARP

If a node wants to send data to a neighbouring node it first needs to know the MAC address of this node. This address can be found using the ARP protocol. The sending node will send an ARP request to find the MAC address of the neighbour. In *ns2* there is a problem that if the ARP request is not answered by

the neighbour (because it was not received, e.g. due to collision) then the message to this node can not be sent. ARP will not retry again and send a second request or report an error if the address can not be resolved. The consequence is that the pending message will stay in the ARP queue forever. As the ARP queue has the length of only one packet, the next time a packet has to be sent to this node, the old packet in the queue is dropped, the new packet is inserted in the queue and a new ARP request is sent. The packet is dropped with the message *DROP_IFQ_ARP_FULL* and in the trace file it is not obvious that this packet has been dropped due to a MAC address that was not resolved for long time. In extreme cases if only one packet is sent to a certain neighbour node and this ARP request can not be resolved, the message will be in the ARP queue until the end of the simulation and then it is dropped with the reason *END*. For the simulations a modified version of the ARP implementation has been used, setting *#define arpOFF*in the *mac/arp.h* file disables ARP. The simulations have been conducted without ARP.

3.1.1.2. Packets Dropped due to Collision

If two packets are received at the same time, the MAC layer has to decide whether one of these packets can be decoded or not (see Sect. 3.1.4.1). If the packet with higher signal strength can not be decoded both packets are dropped. This event will not be written to the trace file with the argument that in reality the MAC layer would not have been able to decode this packet and gather information from it. This leads to the problem that the trace file can contain an entry that a certain packet has been sent or forwarded but this packet is never received. For evaluation purposes it would make sense that these packets are recorded as dropped with an error message that this occurred due to a collision with corrupt packets as consequence. With the current setup it is not possible to find out how many packets have been dropped due to a collision and corrupted packets as consequence. Only the collisions where one packet is received error free can be counted. As it can be of interest how much the MANET suffers from interferences the above mentioned configuration may be not useful.

3.1.2. Scheduling Functions

If a data packet has to be sent, the *forward()* function needs to find a path to the destination node. Therefore it calls *path_find(destination, this_node)*. If the current node is the source of the packet, the scheduling function *sched_get_path()* is called. If the node is not the source, the first path in the route entry will be returned.
The scheduling is implemented as shown in Fig. 3.2. Each available path is assigned a weight that is proportional to the probability that this specific path will be selected. The weights can be set according to different metrics and they define the type of the scheduling method. If all the weights are equal, the scheduling method is round robin. The weights are recalculated whenever a change in the routing table occurs

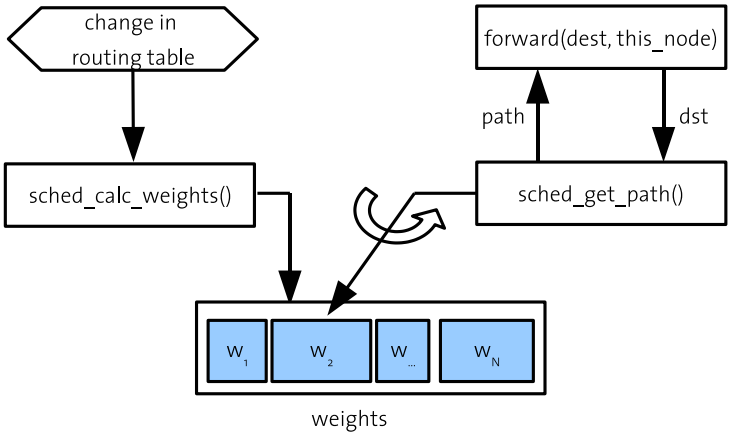

Figure 3.2: Concept of scheduling implementation

and they are stored similar to a cumulative probability function. To select a path the *sched_get_path()* function will choose a random number and search through the *weights* array until the entry is larger than the random number. The corresponding path will then be returned to *forward()* (see Fig. 3.2). To change the behaviour of the scheduling method, the way the weights are assigned has to be changed. The *sched_calc_weights()* function contains a *switch* case to select the different schedulers. This selection is based on a variable that can be passed by the Tcl script.

path	1	2	3	4	5
weights	0	40	40	100	100

Table 3.1: Example for selective Round Robin

In Tab. 3.1 an example of the weights array is given. The second path will be used with probability 0.4 and the fourth path with probability 0.6, the other three paths are not used. The probabilities depend on the selected metric, a simple round robin is implemented and depending on the path length a weighted and a selective weighted round robin is available. The weights are inversely proportional to the length of the path and for the selective WRR, path longer than 1.2 times the average are ignored. One scheduling configuration uses the round trip time (RTT) to select the weights of weighted round robin and one configuration is based on the total number of neighbours seen along the path (select the path with the lowest total number of neighbours). The last two methods depend on information about the path and therefore the weights have to be updated if new information is available. The information is provided using a *Ping* message that is sent along the path.

3.1.3. Additional Packets

3.1.3.1. *Ping* Packet

To gather information of a specific route a *Ping* packet has been introduced. This packet is a special routing packet that is sent over all the available paths from the source to the destination and then returned to the sender. The packet is a routing packet to ensure that it is forwarded with priority in the queues. This is of importance if the round trip time (RTT) shall be measured, otherwise the length of the queue would be part of the round trip time, an effect that we wanted to avoid as we assume that the variance of the measurements would be higher in that case. The packet can measure the RTT of a path and it can collect information about the total number of neighbours along the route. It is extendable to measure other metrics.

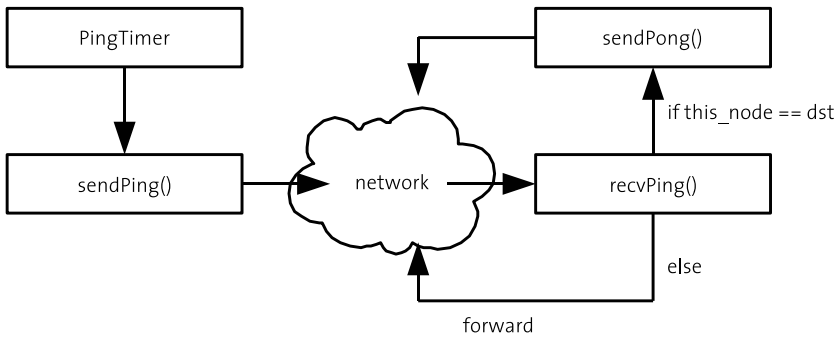

Figure 3.3: Concept of the *Ping* message

The *Ping* message is triggered by a timer as can be seen in Fig. 3.3. After the interval *PING_INTERVAL* a new *Ping* message is sent. If the next hop receives (*recvPing ()*), this message it checks whether it is the destination of the *Ping* message or not. If it is not, it will forward the *Ping* according to its own routing table. If the node is the destination it will call *sendPong()*, this function will create a new packet with

the information to send the packet to the sender of the *Ping* message. The received packet will then be dropped.

The *Ping* packet does not require large changes in the protocol, the packet structure has to be defined in *aodv_packet.h*, a new timer that invokes the *sendPing ()* function has to be created and the functions to send and receive a *Ping* packet have to be provided. The interval to generate a packet can be defined, but the performance of the protocol using the *Ping* packet is much worse than without the *Ping* packet, which can be seen by comparing the two graphs in Fig. 3.4. In the simulation the *Ping* interval has been set to 4 seconds, depending on the packet rate this may be a significant part of the traffic and therefore the routing overhead is increased by this packet. Setting the interval is therefore a trade-off between performance and freshness of the information.
As *Ping* messages shall only be sent along a used route, each node has to find out if it is the origin of a connection and the corresponding destination. This implies that each node has to store a list with the destination entries of its own connections, as only then a *Ping* shall be created. As our traffic generator assumes that only one connection can start from a specific node, we simplified the problem by creating a variable that will store the destination if the current node is the source. This does only work if the assumption, that each node can start one connection, is valid.

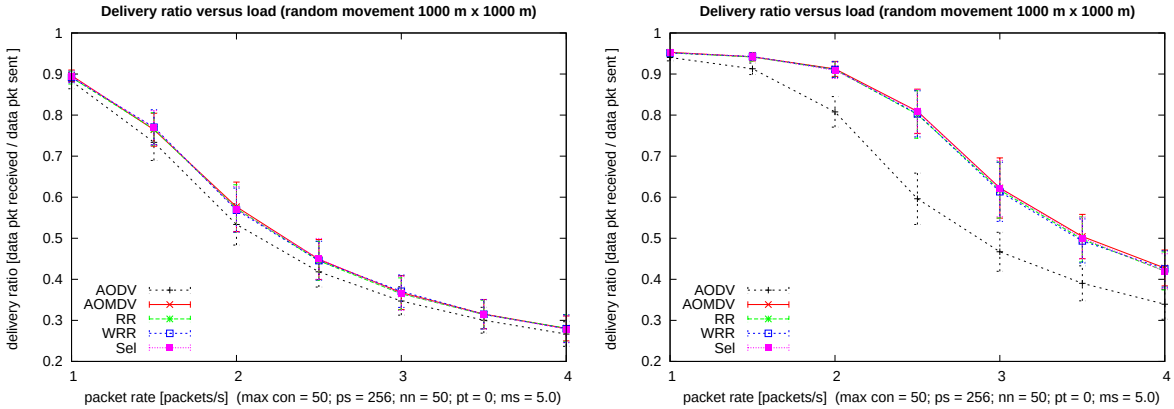

Figure 3.4: Performance of a random movement scenario with and without *Ping* messages

3.1.3.2. *Info* Packet

To have more information available for the analysis than the original trace file, an *Info* packet has been introduced. It can write path information to the trace file whenever a path change has occurred. If all information is written into one file the evaluation is simpler and therefore only one file has been used, even if this method is not very elegant. It would be possible to set a log target in the Tcl file and then use this file to store information. Printing all information into the same file has the advantage that the order of the events is preserved even if multiple events have the same timestamp due to the given resolution.
The *Info* packet is a routing packet but it is never sent over the network, the packet is initialised and then dropped, the information will then be passed to the trace function and will be printed to the trace file.

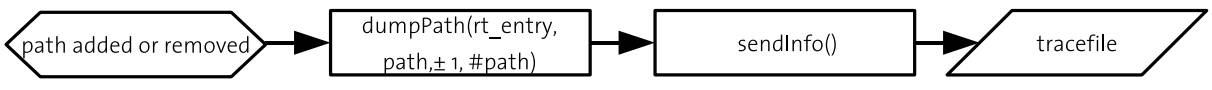

Figure 3.5: Concept of the *Info* packet

The *Info* packet requires a function that can be called if a change in the routing table has occurred. This

dumpPath(rt_entry, path, ±, #path) function takes the routing entry, the path and the number of paths as arguments together with the event that occurred (adding or removing a path). The function stores the information in a struct and calls *sendInfo()*. This is the function that uses the information out of the struct and creates an *Info* packet. This packet is then dropped. This event will invoke a trace function and a corresponding action (in *trace/cmu-trace.cc*) is executed in order to write the information to the trace file (see Lst. 3.1).

```
case AODVTYPE_INFO:
    // dont use the offset to overwrite default printouts
    hdr_aodv_info *info;
    info = HDR_AODV_INFO(p);
    sprintf(pt_->buffer(),  "i -t %f -Hs %d -Hd %d -Ia %d -Ic %d  "
                            "-In %d -Il %d -Ih %d -Ir %f",
    Scheduler::instance().clock(), info->src,  info->dst, info->action,
    info->path_count, info->next_hop, info->last_hop, info->hop_count, info->rtt);
    /* Hs Src, Hd Dest, Ia Action, Ic Path Counter,
     * In Next Hop, Il Last Hop, Ih Hop Count, Ir RTT */
break;
```

Listing 3.1: Handling of the *Info* packet in *cmu-trace.cc*

3.1.4. Protocol Configuration, Selection of Parameters

The simulations were performed using version 2.26 of the network simulator *ns2* [9] with an extension of the AOMDV protocol from S. Das and M. Marina [10]. If not explicitly stated, default parameters of the simulator have been used.

For the initial simulations and the validation of the system the following parameters have been chosen:

dimensions	1km · 1km
number of nodes	50
simulation time	1000s
source type	CBR
max number of connections	30
packet size	256
queue length	50

Table 3.2: Parameter configuration for validation simulations

```
Phy/WirelessPhy set CSThresh_    3.08319e-11 ;# 550m
Phy/WirelessPhy set RXThresh_    1.49226e-10 ;# 250m
Phy/WirelessPhy set bandwidth_   2e6         ;# outdated
Phy/WirelessPhy set Pt_          0.1         ;# 1mW
Phy/WirelessPhy set freq_        2.472e9     ;# 2.472GHz
Phy/WirelessPhy set L_           1.0         ;# system loss
Mac/802_11       set basicRate_  2e6         ;#
Mac/802_11       set dataRate_   54e6        ;# 54Mps
```

Listing 3.2: Parameters to configure wireless behaviour

The parameters in Tab. 3.3 are critical for simulations.

Parameter	Description	Formula	Value
Simulation area	area of topology	$w \cdot h$	1km^2
Node Density	node density in the sim. area	$\frac{n}{w \cdot h}$	50/km^2
Node Coverage	area covered by a node transmission	$\pi \cdot r^2$	0.19km^2
Footprint	area covered by a node transmission in %	$\frac{\pi \cdot r^2}{w \cdot h}$	19%
Maximum Path	max. distance of two nodes	$\sqrt{w^2 + h^2}$	1.4km
Network Diameter	min. number of hops of the max. path	$\frac{\sqrt{w^2 + h^2}}{r}$	6 hops
Neighbour Count	number of neighbour nodes (without edge effects)	$\frac{\pi \cdot r^2}{\frac{w \cdot h}{n}}$	9.81

Table 3.3: Configuration of the validation simulation

```
#ifdef AOMDV
#define AOMDV_PACKET_SALVAGING
#define AOMDV_MAX_SALVAGE_COUNT       10
#endif // AOMDV

#define AODV_EXPANDING_RING_SEARCH
/*
 Allows local repair of routes
*/
//#define AODV_LOCAL_REPAIR

/*
 Allows AODV to use link-layer (802.11) feedback in determining when
 links are up/down.
*/
#define AODV_LINK_LAYER_DETECTION
#define AODV_HELLO

// [scheduling] ---
// #define AODV_PING
#define AODV_INFO
// #define SA_DEBUG
// [scheduling] --- end
/*
 Causes AODV to apply a "smoothing" function to the link layer feedback
 that is generated by 802.11. In essence, it requires that RT_MAX_ERROR
 errors occurs within a window of RT_MAX_ERROR_TIME before the link
 is considered bad.
*/
#define AODV_USE_LL_METRIC

/*
 Only applies if AODV_USE_LL_METRIC is defined.
 Causes AODV to apply omniscient knowledge to the feedback received
 from 802.11. This may be flawed, because it does not account for
 congestion.
*/
//#define AODV_USE_GOD_FEEDBACK

class AODV;
#define MY_ROUTE_TIMEOUT      10   // 10 seconds
#define ACTIVE_ROUTE_TIMEOUT 10 // 10 seconds
#define REV_ROUTE_LIFE        6    // 6 seconds
#define BCAST_ID_SAVE         6    // 6 seconds

// [scheduling] ---
```

```
// we need a timeout for each path, set it the same as the route timeout
#define ACTIVE_PATH_TIMEOUT 10  // 10 seconds
// [scheduling] ---- end

// No. of times to do network-wide search before timing out for
// MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES        3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT    10.0 // sec

// Should be set by the user using best guess (conservative)
#define NETWORK_DIAMETER    30  // 30 hops

/* Various constants used for the expanding ring search */
#ifdef AODV_EXPANDING_RING_SEARCH
#define TTL_START           5    // 5
#define TTL_INCREMENT       2    // 2
#else // NO EXPANDING RING SEARCH
#define TTL_START       NETWORK_DIAMETER    // 5
#define TTL_INCREMENT   NETWORK_DIAMETER    // 2
#endif // NO EXPANDING RING SEARCH

#define TTL_THRESHOLD       7

// This should be somewhat related to arp timeout
#define NODE_TRAVERSAL_TIME     0.03    // 30 ms
#define LOCAL_REPAIR_WAIT_TIME  0.15    // sec

// Must be larger than the time difference between a node propagates a route
// request and gets the route reply back.

#define RREP_WAIT_TIME      1.0     // sec

#define ID_NOT_FOUND        0x00
#define ID_FOUND            0x01

// The followings are used for the forward() function. Controls pacing.
#define DELAY               1.0     // random delay
#define NO_DELAY            -1.0    // no delay

// think it should be 30 ms
#define ARP_DELAY           0.01    // fixed delay to keep arp happy

#define HELLO_INTERVAL      1   // 1000 ms
#define ALLOWED_HELLO_LOSS  3   // packets
#define BAD_LINK_LIFETIME   3   // 3000 ms
#define MaxHelloInterval    (1.25 * HELLO_INTERVAL)
#define MinHelloInterval    (0.75 * HELLO_INTERVAL)

// [scheduling] ----
#define PING_INTERVAL       4   // 4s
// [scheduling] ---- end
```

Listing 3.3: Parameters to configure *ns2* simulation in *aodv.h*

3.1.4.1. Transmission Range

In *ns2* exist three different parameters to define the transmission and interference range of a node. The node transmits with a certain amount of power that can be define in *Pt*. This power is reduced with the distance according to the selected propagation model (FreeSpace, TwoRayGround, Shadowing). If a receiving node receives a signal with a field strength above the threshold *RXThresh* the packet can be decoded error free. If two packets arrive at the same time the ratio of their respective field strengths has to be larger than *CPThresh* (in dB), then the packet with higher field strength can be received error free. Otherwise both packets are dropped. If a node detects a signal above the threshold *CSThresh* it can not decode the packet but it can find out that the medium is not free. Therefore this determines the carrier sense range. If a

node wants to transmit it has first to check whether it can detect a field strength that is above *CSThresh* and only if this is not the case it is allowed to transmit. The two thresholds *RXThresh* and *CSThresh* can be seen as radii around the node, the first indicates the range a node can successfully send its packets and the second the range a node can detect other transmissions. If the MAC layer decides that a packet is received (according to the thresholds) this packet is always received correctly as the channel does not introduce any bit errors.

3.2. Simulation Scripts

3.2.1. Generate Scenario

To create different topologies a generator (*topology_generator.pl*) has been developed. It can either call the generator of *ns2* (*setdest*) or it can build new topologies. Four types can be generated:

- regular hexagon, 61 nodes
- regular hexagon, additional 19 nodes placed uniformly
- regular hexagon, additional 19 nodes in a small area (hotspot)
- circle with uniformly placed nodes, total 80 nodes

The regular scenarios use a distance of 175 m between nodes and it is possible to add jitter to their position. The number of nodes, the jitter and the distance can be changed, as they are variables in the script.
The topology generator can be started by entering the following command with proper command line options as seen in Lst. 3.4. The *type* defines the type of the topology and can be specified by a number, in Fig. 3.6 examples for type 20, 30 40 and 50 can be seen.

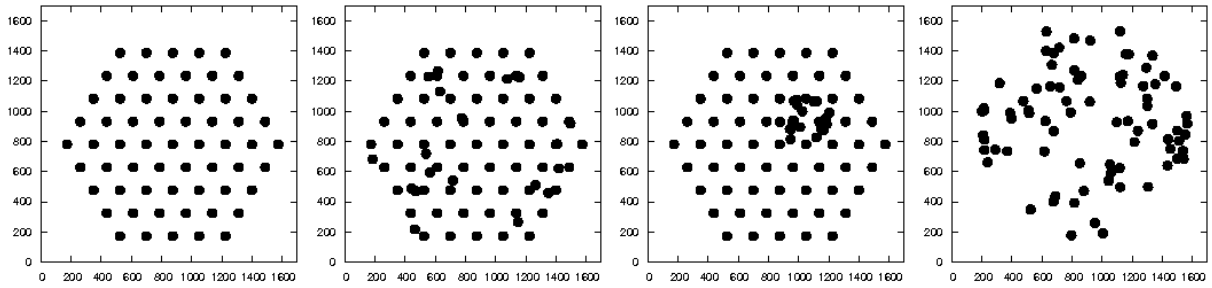

Figure 3.6: Topologies generated by *topology_generator.pl* , type 20, 30, 40 and 50

```
perl topology_generator.pl

use:    -nn [num_of_nodes]
        -pause [pausetime]
        -speed [maxspeed]
        -simtime [simtime]
        -x [maxx]
        -y [maxy]
        -type [type of scen]
        -seed [random seed]
        > [outdir/movement-file]
```

Listing 3.4: Using *topology_generator.pl*

To distribute the traffic uniformly a traffic generator has been created (*traffic_generator.pl*). It can be used as seen in Lst. 3.5. Even if the traffic type has to be given, only *cbr* is implemented, *tcp* will return an error.

```
perl traffic_generator.pl
use:    -type [cbr|tcp]
        -nn [number of nodes]
        -seed [random seed]
        -nc [number of connections]
        -rate [packet/s]
        -pktsize [packet size]
```

Listing 3.5: Using *traffic_generator.pl*

The source and destination are chosen randomly with the restriction that a specific node is not allowed to have more than a maximum number of connections. In the script, the number of how many connections maximally can start and end in each node can be defined. For both generators a random seed can be set to be able to reproduce the state.

3.2.2. Setting Up a Simulation

As the simulation environment has many free parameters a large amount of calculation power is required to find optimal results. As the simulations are independent of each other they can be run simultaneously on different machines. To have more CPU power available *Condor* [11] has been used. *Condor* is a system that manages and distributes jobs on different machines if they have free CPU resources. To simplify the start of a simulation different scripts have been developed.

3.2.2.1. *Condor*

Condor [12] is a scheduler that runs jobs on idle workstations. On the computer infrastructure of the EE department the *Condor* server is installed and all machines hosted by the ISG.EE such as *tardis-d01* and *tik43x* are part of the cluster.
A *Condor* job is defined by an executable file (which is the same for all jobs). The job is started on an idle workstation. If anybody logs on this machine the job is set into sleep mode. If the user is active longer than a certain time the sleeping task is killed. All information is lost and the job has to be restarted on an other machine. Consequential the jobs should have a runtime shorter than one hour to increase the chance that they are not interrupted.
As each job runs on a different host the network file system (NFS) is used to give the jobs access to the home directory. Each job will store its output in this directory and depending on the number of jobs and the size of the output per job, enough disk space is required. To submit the jobs to *Condor* a file (*run_simulation.condor*) containing a list with all jobs is used and the following command will submit the jobs to the cluster.

```
schadomi@tardis-a03:~> condor_submit run_simulation.condor
```

The *Condor* file is constructed by the *create_simulation.pl* script and contains all jobs and system parameters for a *Condor* session. The executable file is the *remote.pl*.
Each job has the rights of the user that submitted the job. The effect is, that as a student with normal ITET rights, the *Condor* jobs can only be executed on the workstation of the ISG student rooms. To access the computers of the TIK as student some changes in the submit file have to be done. With the command

```
+USER_GROUP     = "tik"
+JOB_GROUP      = "tik"
```

in the *Condor* file the rights management of *Condor* is overwritten and the jobs run on machines of the TIK institute and the ISG workstations.
By default for each completed job an email is sent. To avoid this

```
Notification = Error
```

should be placed in the *Condor* file, as then only in the case of an error a notification is sent. The progress and error messages can nevertheless be found in the *log* folder. By defining requirements on the infrastructure specific clients from the cluster can be selected. As the simulation runs only on Linux the following configuration has been used.

```
Requirements = ( ( Arch == "INTEL"
                || Arch == "PPC"
                || Arch == "x86_64"
                || Arch == "ALPHA"
                || Arch == "SUN4u" )
               && (OpSys == "LINUX" )
               && Memory >=32 )
```

3.2.2.2. Create Files Required for Simulation

To create a package with all the files required to start a simulation with *Condor* the *create_simulation.pl* script has been developed. The script can be configured by specifying the parameters as in Lst. 3.6.

```
#--------------------------------------------------------------------------------

# General
my $n_of_run        = 5;            ## number of simulations

# Variables
my @queue_type      = ('Queue/DropTail/PriQueue');  ## interface queue type
my @queue_length    = (50);         ## length of interface queue (in packets)
my @dim_x           = (2000);       ## x (network dim) --> 2000 for scene != 00
my @dim_y           = (2000);       ## y (network dim) --> 2000 for scene != 00
my @n_node          = (80);         ## number of nodes ---> 80 for scene != 00

my @sim_time        = (1000);       ## simulation duration

my @sim_progress    = (100);        ## simulation progress, not used

# Moving pattern
my @pause_time      = (0);          ## pause time (pause between moving)
my @max_speed       = ('0.0');      ## max speed [m/s]
my @scen_type       = ('40');       ## see topology_generator.pl for help

# Traffic pattern
my @source_type     = ('cbr');      ## type of traffic source ("cbr" or "tcp")
my @num_connect     = (50);         ## number of connections
my @packet_size     = (256);        ## packet size [Bytes]

my @packet_rate     = ('1.0','1.5','2.0');  ## packet rate [packet/second]

# AOMDV parameter
my @sched_type      = ('0','1','2','3');    ## type of path scheduler

                                    ## 0 = Select only one [aomdv] (shortest path)
                                    ## 1 = Round Robin [RR]
                                    ## 2 = Weighted RR [WRR] (1/hop)
                                    ## 3 = Selective WRR [SWRR]
                                    ## 4 = WRR RTT (needs ping)
                                    ## 5 = Neighbour (needs ping)
my @max_path        = (1);          ## max of paths between source and destination

#--------------------------------------------------------------------------------
```

Listing 3.6: Configure simulation parameters

According to these parameters the traffic and topology patterns are created (*traffic_generator.pl*, *topology_generator.pl*). Those patterns are stored in the *pattern* folder and they are only generated if they do not yet exist. The script also creates the *Condor* file that is used to sumbit the jobs. Before a job is added to the list, it is checked if the point has already been simulated (simulated results can be found in the *sim_data* folder). With all the required files a *tar* archive is created and copied to a remote machine.

3.2.2.3. Script to Run Simulations on Remote Host

To start the simulation the *remote.pl* script is used. This script takes command line arguments that are then passed via command line to the Tcl script that actually starts the simulation. The script sets the paths to the Tcl libraries and waits until the *.sim* file has a nonzero filesize or a timeout occurs. This check has to be done as different processes are started by the script and it they have to be finished before the script terminates, otherwise the simulation results get lost. Even if the *ns2* executable is compiled using *static* flags the Tcl libraries are installed on one of the remote hosts (and then accessible via NFS) to avoid conflicts. The source to install the libraries can be found in *source/ns-small-2.26.tar.gz*

3.2.2.4. Parameters for *ns2*

The *aomdv.tcl* scripts is the interface to the simulation environment. This script is used to set all parameters of *ns2* and to start the simulation. In Lst. 3.7 the parameters of the Tcl file can be seen.

```
set tracefile_              [lindex $argv  0]; # tracefile
set mobility_pattern_       [lindex $argv  1]; # mobility pattern
set traffic_pattern_        [lindex $argv  2]; # traffic pattern
set queue_type_             [lindex $argv  3]; # [CMUPriQueue Queue/DropTail/PriQueue]
set queue_length_           [lindex $argv  4]; # Queue length max packet length in ifq
set dimension_x_            [lindex $argv  5]; # Dimension X
set dimension_y_            [lindex $argv  6]; # Dimension Y
set number_of_node_         [lindex $argv  7]; #
set sim_time_               [lindex $argv  8]; # Simulation duration
set sim_progress_           [lindex $argv  9]; # Simulation progess = pause_time?
set sched_type_in_          [lindex $argv 10]; # type of AOMDV scheduler
set aomdv_max_path_in_      [lindex $argv 11]; # maximum number of paths
```

Listing 3.7: Parameters in the *aomdv.tcl* file

To evaluate the simulation all the actions of the routing and data packets are written into a trace file. The file is opened in this script and the filehandler is passed to the C functions. The problem with this setup is, that Tcl does not allow files larger than 2GB. This file size is reached if the number of nodes is high or if a lot of route requests (flooding) are sent. For a simulation with 100 nodes the 2GB are reached after a simulation time of about 500 seconds. The script then reports an error and does not write any more data to the file. To circumvent this problem we decided not to write the data to a file but to parse it on the fly. This can be done by piping the output to the used gawk scripts. This solution requires more RAM than writing the data to a file but this has not been a problem with our simulations.

```
set tracefd [open "| gawk -f tr_to_val.awk | \
        gawk -v M_N = $number_of_node_ -f val_to_sim.awk > $tracefile_.sim " w]
```

3.2.2.5. Start a Simulation

To start a simulation follow the following steps.

1. Installation:
   - Install the Tcl libraries in the ITET NFS space
   - Adjust library paths in *remote.pl*
2. Edit *create_simulation.pl*
   - Adjust system environment parameters as path to *ns2* , username
   - Adjust simulation parameters such as number of nodes, packet rate
   - Adjust simulation version and testseries tags.
   - `perl create_simulation.pl`
3. Start condor

- Log in on an ISG machine
- Change to the packet folder, e.g. *cd /home/schadomi/extra/V001-001*
- Submit condor job list by *condor_submit run_simulation.condor*

4. Collect results
   - wait until all simulation are finished
   - tar the simulation results of the *sim_data* folder with *tar czvf sim_data_XXX.tar.gz sim_data*
   - copy the archive to the localhost with *scp*
   - extract archive into the local *sim_data folder*

3.2.3. Condense Information

3.2.3.1. Reduce Size of Output

The *tr_to_val.awk* is used to save the information of the trace file in a more compact form. This is achieved by removing unnecessary information. The script has been developed by Rainer Baumann and was modified to support *Ping* and *Info* packets.
During our simulation a bug in *tr_to_val.awk* was found if a node with IP address zero is used. In reality this zero is not used for nodes (broadcast) but in the simulation this can be the case. If the topology generator of *ns2* is used, the input of 50 nodes returns nodes with id's from 0 to 49. The problem is that the topology generator begins to number the nodes with zero. As consequence the node with id zero will be created. This bug has been fixed.

3.2.3.2. Collect Data for Analysis

The *val_to_sim.awk* script uses the output of the *tr_to_val.awk* script as input to collect the needed information for the analysis and stores it in a *.sim* file. These files have unique file names based on the simulation parameters. The script is based on an existing script that collects data for each node.
In the first 180 seconds of the simulation all connections are set up and it is assumed (without proof) that after this time the network is in steady state. As the start up time should not be taken into account, only packets that arrive after the first 180 seconds are counted.
The *val_to_sim.awk* scripts consists of different parts, one for the initialisation, one to handle the different events and the last is performed at the end.
Five different events are distinguished. Either a packet is sent, forwarded, dropped or received or an *Info* packet has been printed to the trace file. All these events will trigger different actions, if a packet has been handled, different counters are incremented, the *Info* packet will be parsed and the information is stored in different tables. Handling an *Info* packet is quite different to the other events and therefore it will be explained in more detail. In a first step the routing tables of all nodes are rebuilt or updated using the information of the *Info* packet and these tables are stored in one large table. If a change in the routing table of a sending node occurs all the paths from this source to the destination are reconstructed. As all the information of the nodes routing tables is available a recursive function searches through these tables and constructs all possible paths to the destination without a check for disjointness. The found paths represent the state of the routing tables at this time instance. Packets are forwarded by looking up the routing entry for the corresponding destination and therefore all the paths that the protocol may use can be found with the recursive function. The result of this recursive search is stored in a table together with the information at which time this path has been existing in the network. These information can be used to reconstruct the routing tables at any given time instance.
To count the number of disjoint paths from source to destination other information is required. The reconstructed paths are not disjoint but in the *Info* packet the number of disjoint paths for each connection is indicated (as the source only knows about disjoint paths). This values are used for the statistics of the number of available paths.

4. Evaluation Tools

To show the behaviour of the network, different evaluation tools have been developed. Either one specific simulation run can be investigated in detail or statistics over multiple runs can be created. These tools generate graphical output to visualise the results. As we used a special scenario (hexagon with hotspot) a topology generator was created and to distribute the traffic uniformly on the nodes a traffic generator had to be developed.

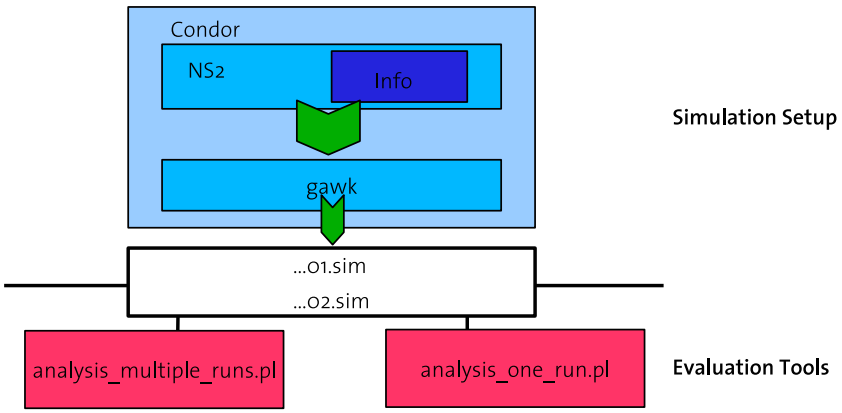

Figure 4.1: Structure of the simulation environment

4.1. Evaluate One Simulation Run

To evaluate different aspects of the protocol, one specific run can be investigated. For scheduling the number of paths available at a given time instance as well as average over the simulation is of interst. To determine whether the implementation is correct the content of the routing table is used to reconstruct all possible paths and to identify the bottleneck of the simulation the dropped packets are plotted over the simulated area. All these evaluations are done for one specific simulation and are implemented in the perl script *analysis_one_run.pl*. The script reads the data from the *.sim* file that is passed as command line argument (see Lst. 4.1) and calculates all the required values in advance, then a menu is displayed to select different functions.

```
perl analysis_one_run.pl
use: -file [SIM FILE]
```

Listing 4.1: Using *analysis_one_run.pl*

4.1.1. Temporal Behaviour of Number of Paths

The number of available paths at a given time instance is very interesting for the scheduling. As in Fig. 4.1.1 the number of available paths for one connection can be plotted. The script also calculates the average number of paths for this connection over the entire simulation time. If no paths are available the routing protocol will generate a RREQ to find a route to the destination.

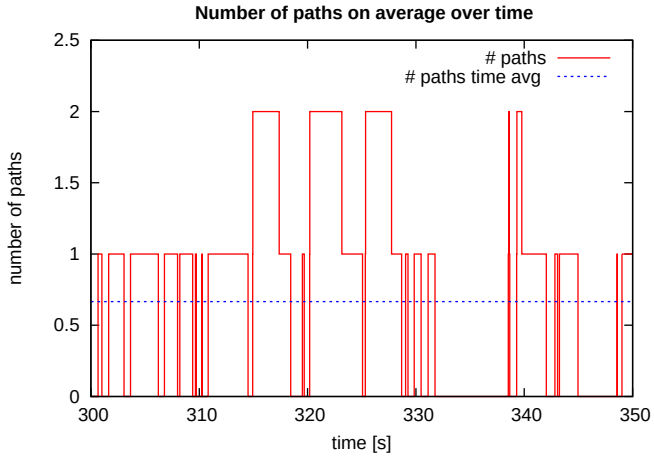

Figure 4.2: Number of paths plotted over a time interval

4.1.2. Effectiveness of the Flooding

'Averaging' the average number of available paths over all the connections with the same length (length of shortest path) is plotted in Fig. 4.3. To see the performance of the flooding the average number of paths found by a route request can be plotted. A route request is sent if no path is available therefore the route requests can be identified by the zero periods as seen in Fig. 4.1.1. The maximal value between two zero events is the maximal number of paths found by a route request. The number of connection is also plotted as this is a measure of how many samples are available. To show the accuracy of the average values the standard deviation is plotted.

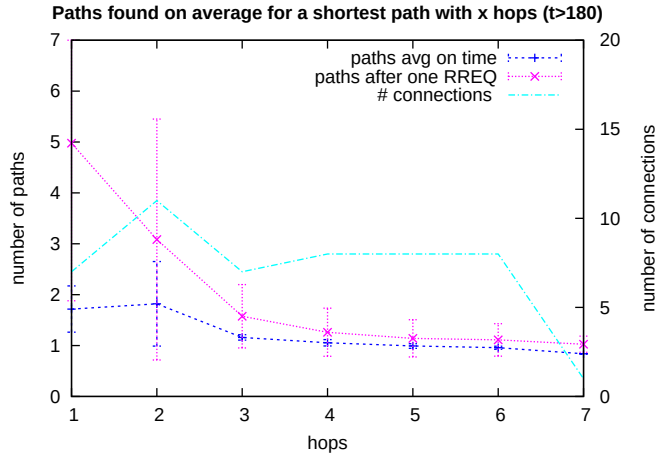

Figure 4.3: Number of paths found on average partitioned by connection length

4.1.3. Visualise Routing Table

The *Info* packets contain all the information that is required to reconstruct the routing table at any given time instance. These information is collected by the script *val_to_sim.awk* and the routing table is created. In theory the protocol claims that it finds node or link disjoint paths. But by rebuilding the routing table it can be shown that non-disjoint paths can be created. Each node knows the path to a destination together with the next and last hop. At a forwarding node the last hop of the selected path is not taken into account and therefore the paths can not be guaranteed to be disjoint. In the routing table leftovers from the flooding can be found, as reverse paths are inserted in the table and will expire after a certain time; see Sect. 5.2.4.1. The *analysis_one_run.pl* script can draw the state of the routing table at a given time instance. It can create snapshots of these states and build an animated gif with correct timing between the snapshots. To create the animated gif a small program named *whirlgif* [13] has been used.

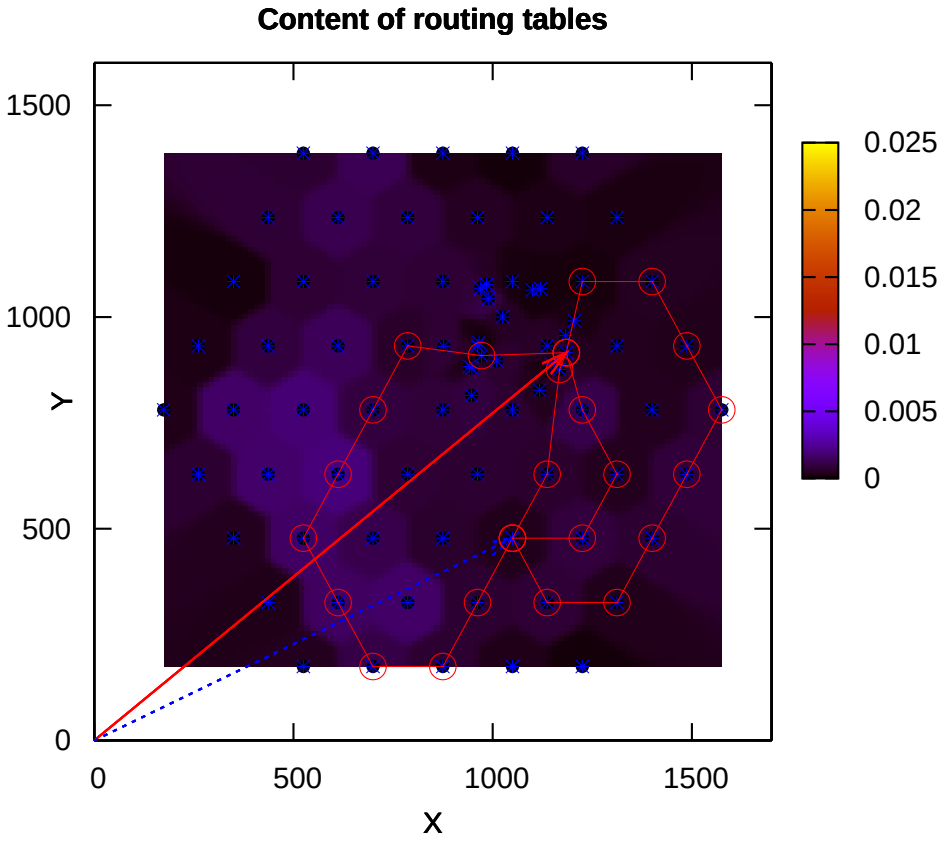

Figure 4.4: Content of the routing tables

4.1.4. Area with High Rate of Dropped Packets

The script *analysis_one_run.pl* can create a plot of the nodes and draw the relative amount of dropped packets. Gnuplot has been used to plot the area and interpolate the values between the nodes. To be able to use this script information about the ratio of dropped packets is required which is calculated in *val_to_sim.awk*. Drawing the dropped packet depending on the position if the node makes only sense if the scenario is static. For a mobile scenario a solution could be to record the coordinates of each dropped packet and to create then a map out of this data. But as the tool has only been used for static scenarios the packets are assigned to a node and not a position. Figure 4.5 shows an example of the dropped packets plotted over the simulated area.

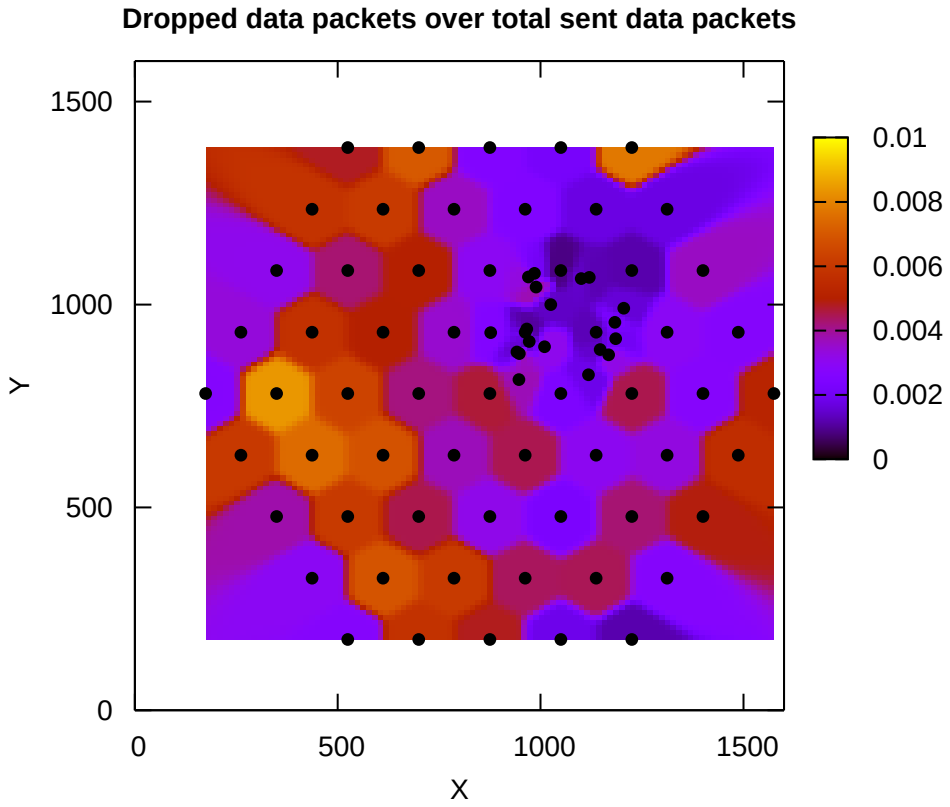

Figure 4.5: Dropped Packets

4.2. Evaluate Multiple Simulation Runs

The evaluation of multiple simulation files is done with the scripts *analysis_multiple_runs.pl*. The script expects that all the files to evaluate are stored in one folder. The script then uses a filtering function (based on *ls* and *grep*, see Lst. 4.2) to find the files that are part of the analysis. The filter is based on the fact that all the information to identify a simulation is part of the filename. Listing 4.3 shows how to set the filter parameters.

```
for($x_dim_index= 0; $x_dim_index<scalar(@x_dim_grep); $x_dim_index++)
```

Listing 4.2: Use *grep* and *ls* for filtering

```
# set fix parameters
#--------------------
# Version eg V002       ="V001.*"
# number_of_run         ="ri-.*";
# queue_type            ="qt-0-.*";
# queue_length          ="ql-50-.*";
# dimension_x           ="dx-1500-.*";
# dimension_y           ="dy-1500-.*";
# simulation_duration   ="sd-1000-.*";
# sim_progress          ="sp-10-.*";
# source_type           ="st.*cbr.*";
# ending                ="sim";

my $queue_type          = 0;
my $queue_length        = 50;
my $dimension_x         = 2000;
my $dimension_y         = 2000;
my $sim_duration        = 1000;
my $sim_progress        = 100;
my $source_type         = 'cbr';
elsif($choice_fix eq "hexa hotspot")
{   #scenario 40

    $dimension_x            = 2000;
    $dimension_y            = 2000;


    $fix_grep = "$version".
                ".*ri".
                "-.*qt-$queue_type".
                "-.*ql-$queue_length".
                "-.*dx-$dimension_x".
                "-.*dy-$dimension_y".
                "-.*sd-$sim_duration".
                "-.*sp-$sim_progress".
                "-.*scp-40".
                "-.*st.*$source_type".
                ".*$simulation_result_type";
    $fix_title = "hexagon hotspot $dimension_x m x $dimension_y m";
    $fix_file_name = "[$version-dx-$dimension_x-dy-$dimension_y-hexa-hotspot]";
}
```

Listing 4.3: Filter of *analysis_multiple_runs.pl*

To select the values that are plotted, in Lst. 4.4 for each axis the properties can be set (the z-dimension are the protocol/scheduler configurations). The configuration of the example will result in a plot of the load of AODV, AOMDV, RR, WRR and selective WRR versus the delivery ratio.

```
$choice_z   = 'only4';
#$choice_z  = 'all';

$choice_x   = 'load';
#$choice_x  = 'load_fine';
#$choice_x  = 'node density';
#$choice_x  = 'movement pause time';
#$choice_x  = 'movement speed';

$choice_y   = 'ratio';
#$choice_y  = 'delay';
```

Listing 4.4: Selection of the axes

The output of the *analysis_multiple_runs.pl* script is an *.eps* file that is stored in the *temp* folder. In Fig. 4.6 an example of the output of the *analysis_multiple_runs.pl* script can be seen.

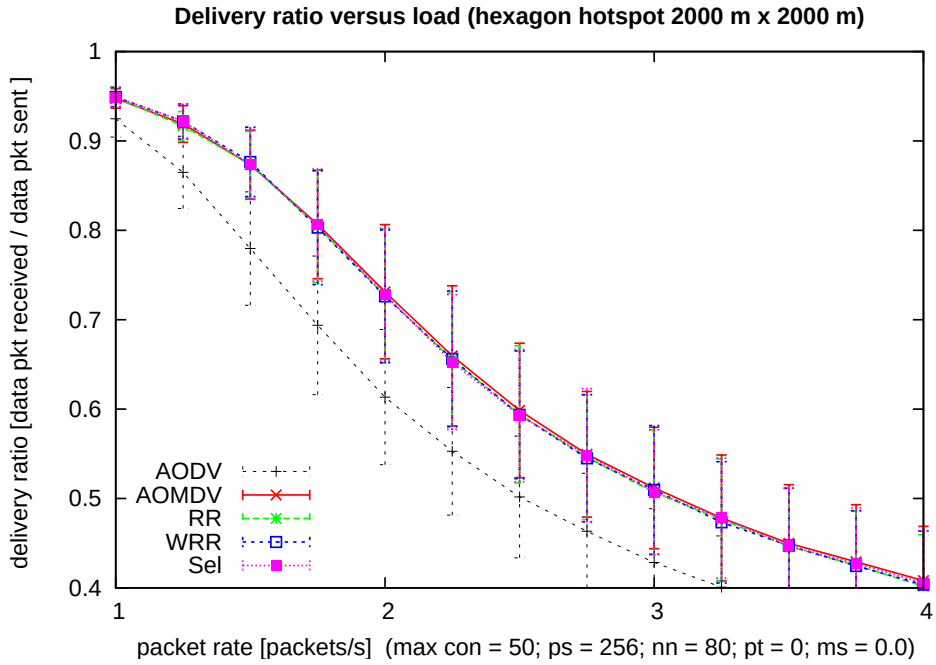

Figure 4.6: Example of the output of *analysis_multiple_runs.pl*

5. Simulation Results

5.1. Validation of the Environment

To ensure that the simulation environment behaves as expected, different simulations have been conducted. It can be seen that the AOMDV protocol using multiple paths has the same or better performance as the AOMDV with only one path (AOMDV as singlepath is similar to AODV), a result that has been expected. The different scheduling algorithms that have been implemented do not show a general trend, on average they have the same performance as the AOMDV with multiple paths. All the simulations have been conducted using node disjoint paths even if there exist more link disjoint than node disjoint paths. Using link disjoint paths is problematic as the implementation of the protocol can not determine which input path corresponds to which output if a node is part of multiple link disjoint paths. If the scheduling uses information about a specific path it is necessary that the packet is sent via this defined path, but the protocol can not guarantee this. To avoid this problem node disjoint paths have been used.

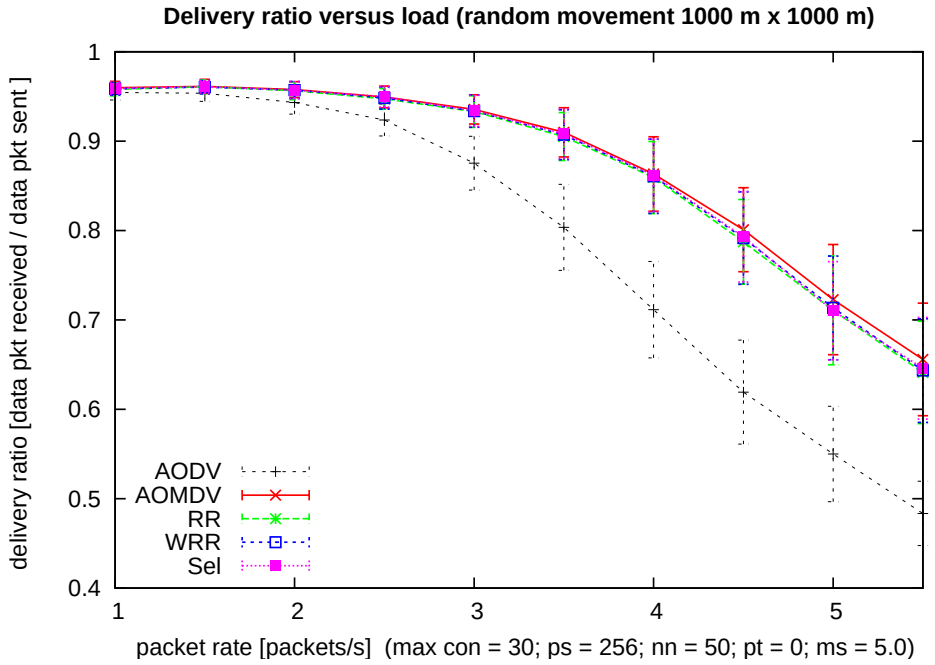

Figure 5.1: Throughput of a random mobility, random placement scenario

5.2. Static Hexagonal Scenario

As the results from the validation did not show large differences for the different schedulers, we decided to create a smaller problem where a certain question can be investigated. The problem was based on a static scenario and uniformly distributed traffic. The question was if it would be possible to increase the performance of a hospot scenario (certain area with high node density) by using the node density as a scheduling metric. To answer this question a hexagon with equidistant nodes has been chosen and some nodes have been placed randomly in a small area to create a hotspot (for the topology generator refer to Sect. 3.2.1). To investigate this problem it has been splitted into different subproblems. As the throughput shall be increased first the bottleneck of the scenario has to be identified. Then the question is if enough paths can be found to allow scheduling. The last question is if always the same paths are found and where in the scenario they are located. As the throughput should be increased the paths have to allow a routing around the bottleneck. In Fig. 5.2 one realisation of a hexagonal hotspot scenario can be seen.

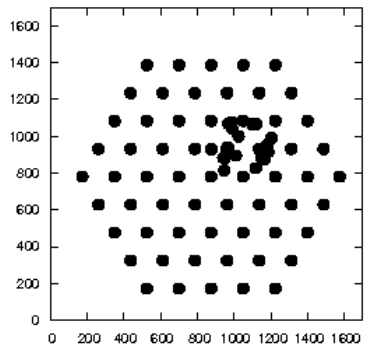

Figure 5.2: Hexagonal hotspot scenario

Figure 5.3 shows the throughput of a hexagonal hotspot scenario. It can be seen that the different scheduling methods do not increase the performance. As expected the throughput of the multipath configuration is higher than with the singlepath configuration.

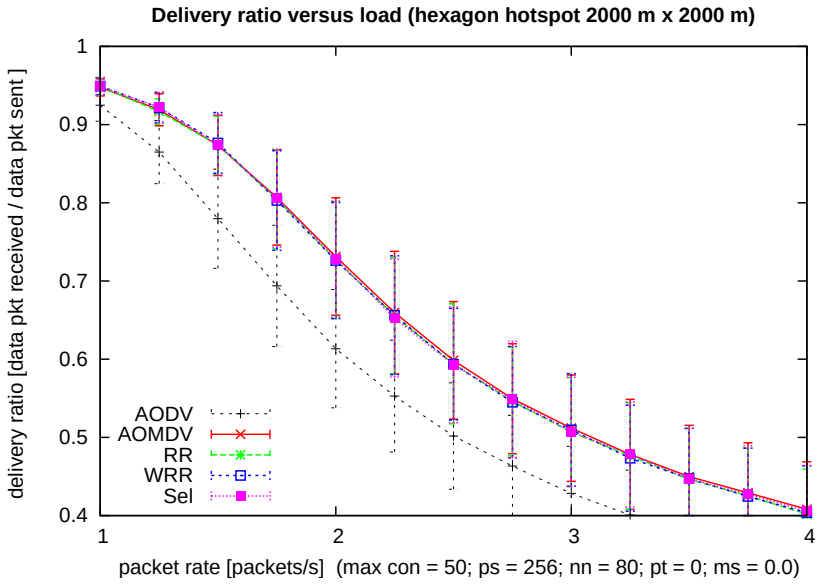

Figure 5.3: Throughput of a hexagonal hotspot scenario

The following configuration has been used for the simulations to answer these questions:

dimensions	2km · 2km
number of nodes	80
simulation time	1000s
source type	CBR
max number of connections	50
packet size	256
queue length	50

Table 5.1: Parameter configuration for the simulation with hexagonal scenario

The following parameters are critical for simulations:

Parameter	**Description**	**Value**
Simulation area	area of hexagon	1,98 km^2
Node Density	node density	40/km^2
Node Coverage	area covered by a node transmission	0.19km^2
Footprint	area covered by a node transmission in %	9,6%
Maximum Path	max. distance in hexagon	1.75km
Network Diameter	max. shortest path through hexagon	10 hops
Neighbour Count	number of neighbour nodes	>6

Table 5.2: Settings of the hexagon simulation

5.2.1. Number of Paths over Time

The number of paths that are available at a given time instance can be plotted using *analysis_one_run.pl* (see Sect. 4.1.1). For the hexagonal hotspot scenario the results are given in Fig. 5.4 for two different loads. If no path is available a route request is sent and the found paths are inserted. In Fig. 5.4 the same time interval and the same connection is plotted, only the packet rate is changed. In both plots periods without available paths can be observed. As the scenario is static, paths can not get lost due to mobility. The paths are lost as link layer detection is used. This means that the link layer will try to send a packet during a given interval. If it is not able to transmit the packet within this time, it assumes that the link is broken and reports a link failure to the routing protocol. The routing protocol will remove this link. There exist two different reasons why a packet can not be sent to the next node. The first is if the node is out of reach due to mobility, the second is due to congestion. If the wireless medium is busy the link layer has to wait until it can transmit the packet. Depending on the queue length either the packet is dropped due to queue overflow or the link layer reports an error due to the timeout. If a node is in a congested area it is possible that all its paths to a destination are congested and they will be removed due to a reported link failure. A new route request has to be sent (increasing the load) and again only congested links can be found (as there exist only congested paths). Therefore the link layer detection may decrease the performance of the protocol as it is not possible to distinguish between broken and congested links. The higher the load in the network the higher the chance that links are congested and get removed by the routing protocol. This can be observed in Fig. 5.4. In the plot with higher load more route requests have to be sent compared to the plot with lower load. The average number of available paths is smaller if the load is higher, again due to the link layer detection.

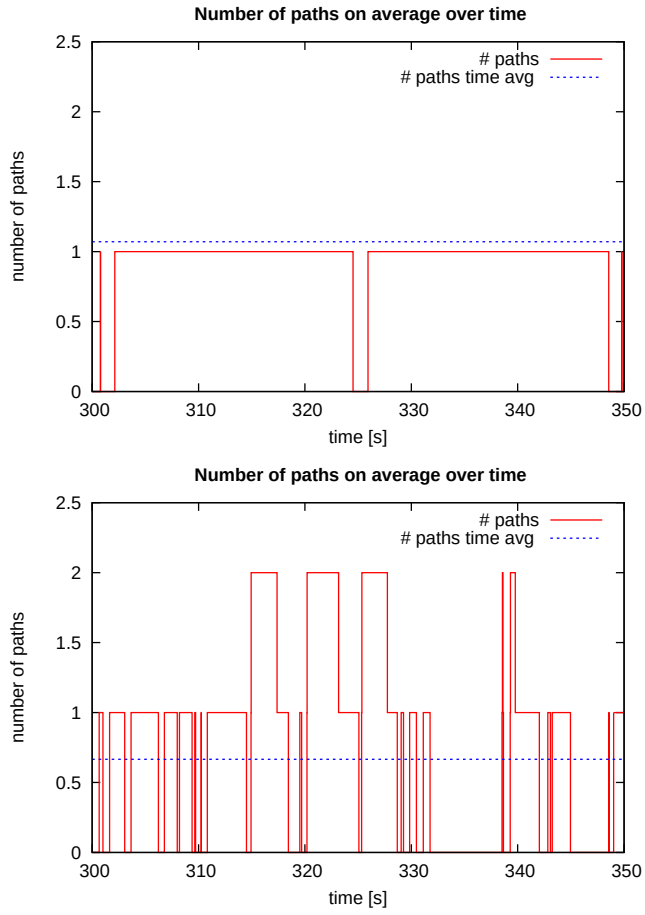

Figure 5.4: Number of paths for a time interval, 1 and 2 packets/s per connection

5.2.2. Number of Paths on Average

As scheduling makes only sense if it is possible to choose from more than one path, it is important to find out how many paths the protocol can really find. For the hexagonal hotspot scenario the number of paths found for longer distances approaches one. Even for paths with three hops, the average is below two (on average not always a choice) and consequentially, with this flooding method, scheduling makes sense only for paths with maximal three hops. The idea was to increase the throughput by routing around the bottleneck of the scenario, however this is not possible with three hops. Therefore the protocol is not suited for the given problem. It can also be seen that for higher load the number of paths found by a RREQ is lower as for a smaller load. The reason can be that more collisions can occur and the RREQ get lost or that they might be dropped due to full queues. The consequence is that less RREQ will reach the destination and less paths will be found.

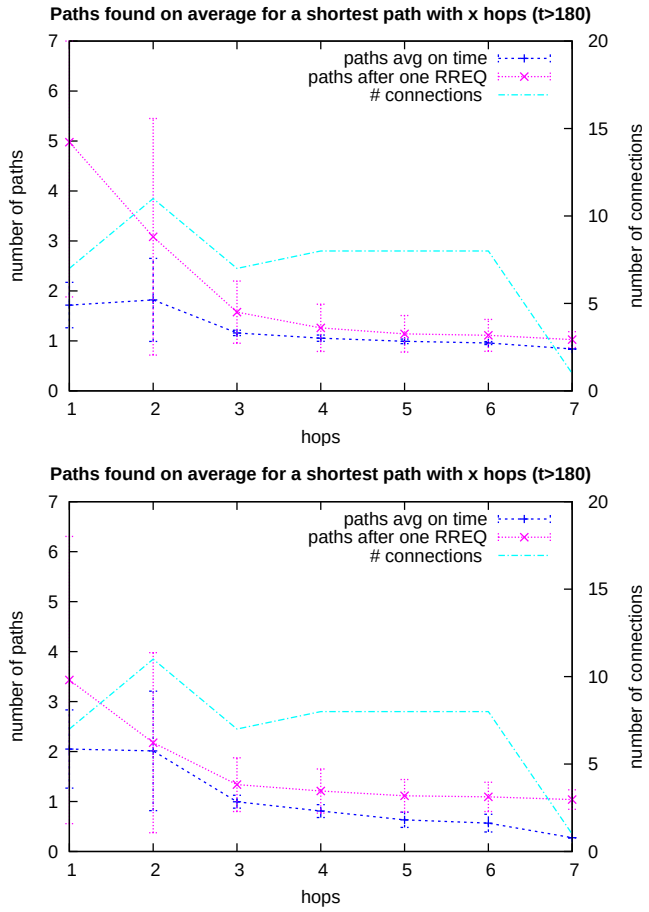

Figure 5.5: Number of paths ordered with shortest path length, 1 and 2 packets/sec

Even if a static scenario is used path can get lost (Sect. 4.1.1). The average number of paths depends on the load of the network. In Fig. 5.5 the average number of paths available during a simulation is plotted for two different loads. For higher load the average is lower and for paths with four hops or more the average is below one, this means that not at any time a path exists to the destination. The average number of available paths can be seen as a measure of the connectivity of the network. If no paths to a destination are available a RREQ has to be sent. The more often a route is deleted the more requests are sent. This results in an increased load of the network which will decrease the number of available paths.
To find node or link disjoint paths, all first and last hops have to be unique and hence the number of neighbours defines the number of disjoint paths that can exist. In Fig. 5.6 an example of the flooding is plotted

for a hotspot scenario. All the nodes that broadcast the request that has passed a specific first hop are coloured with the same colour. If a connection is constructed from source *S* to destination *D1* only one path will be found, as *D1* receives only RREQ that have been forwarded via node one. If a connection from *S* to *D2* is created, two paths can be found as RREQ's via node one and node two are received. Obviously the chance is small that *D2* will ever receive a RREQ that has been forwarded from node five.

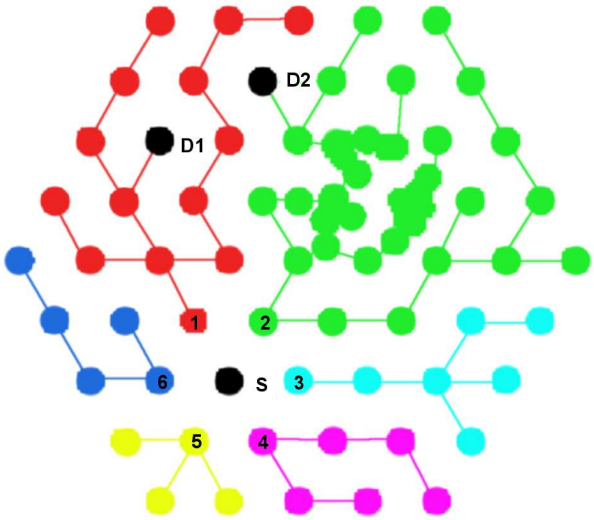

Figure 5.6: Concept of the flooding mechanism

5.2.3. Number of Dropped Packets Plotted as Area

As the idea is to increase the throughput, the bottleneck of the simulation has to be identified. Therefore the number of dropped packets divided by the total number of sent packets is plotted in the 2D area of the simulation. In Fig. 5.7 the results for the hotspot scenario with a load of 1, 2 and 4 packets/sec are illustrated. Light colours correspond to a high number of dropped packets, darker colours represent low numbers of dropped packets. The 50 connections are uniformly distributed over the nodes and hence the chance that many connections start and end in the hotspot is high. It can be seen that the hotspot is not the area with the highest probability that a packet is dropped. It can be concluded that the hotspot is not the bottleneck of the scenario but any further conclusions about the bottleneck would require new simulations to exclude any edge effects.

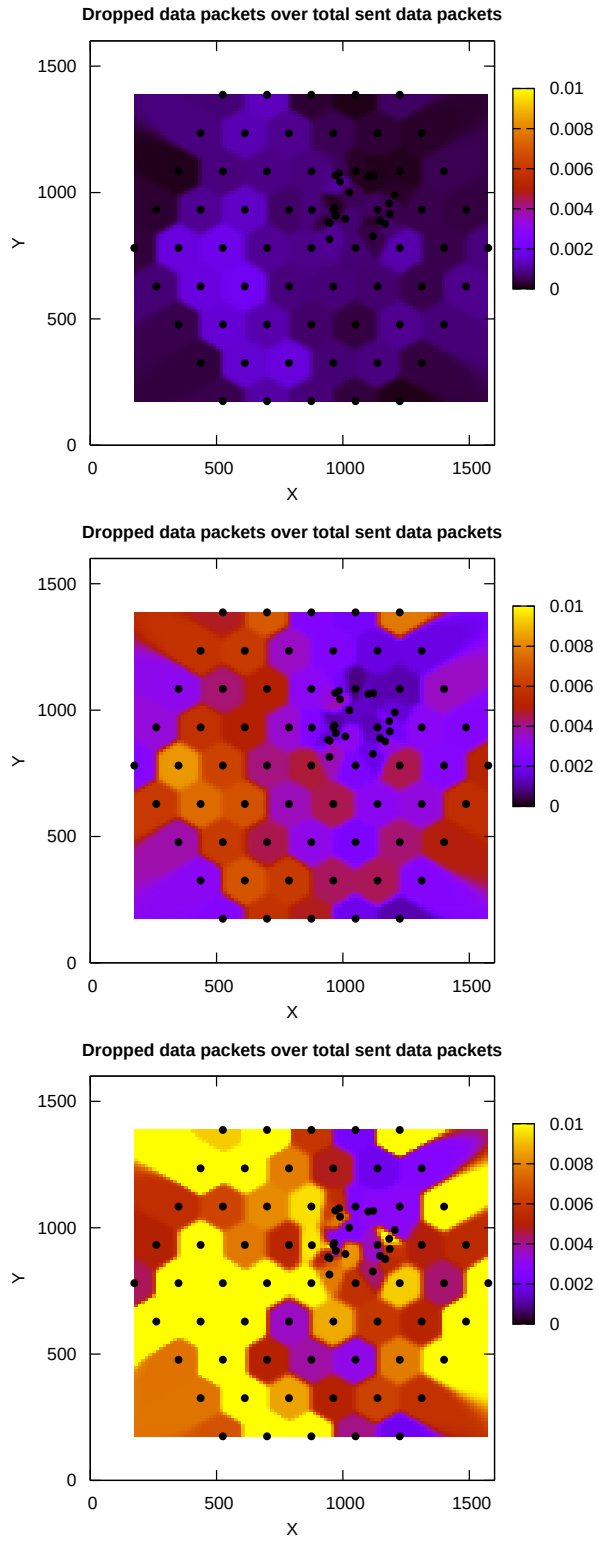

Figure 5.7: Dropped packets plotted in a 2D area, 1, 2, and 4 packets/sec

5.2.4. Routing Table

As the routing protocol has a large impact on the performance, it is interesting to see the behaviour of the protocol over time Sect. 4.1.3. In Fig. 5.8 the content of the routing tables has been used to construct all possible paths from source (red arrow) to destination (blue arrow). The destination can be reached with three hops and at this time instance four paths can be found.

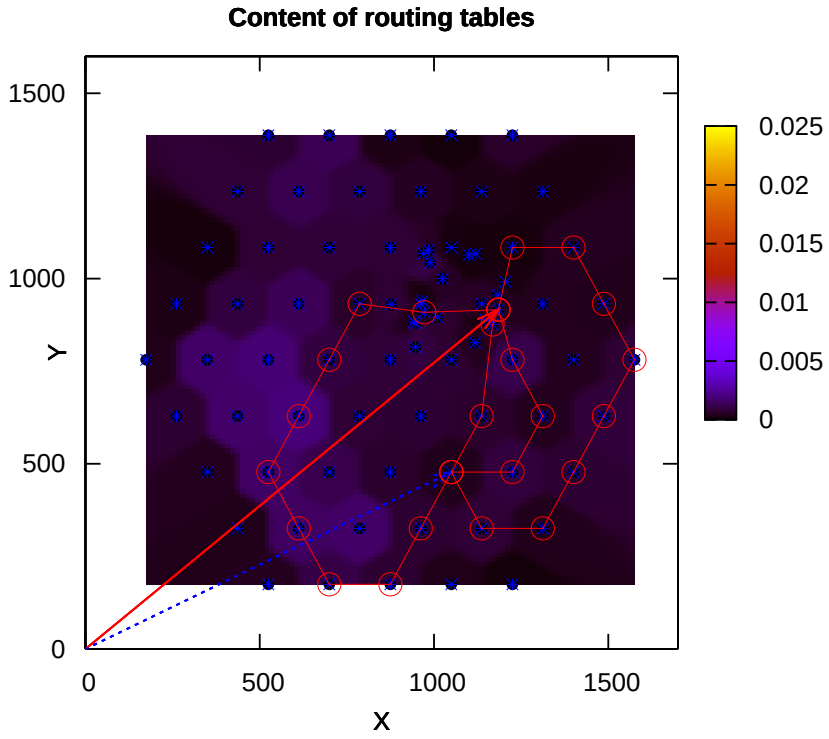

Figure 5.8: Content of the routing tables at a given time instant

5.2.4.1. Run Condition of AOMDV

AOMDV uses flooding to set up a route from a source to a destination. All nodes that receive a RREQ store the information from which node the request has been sent and via which node it was received. The route replies will then be forwarded to the source of the request using this information. Each node decides if the received information will be used to add a new disjoint path. The backward paths are not removed, they only get a lifetime that is shorter than the lifetime of an active route (but they will live for 6 seconds). Until these path are expired, the routing table can contain entries that are not part of a disjoint path. As forwarding nodes select the first available path, without checking if the last hop is the one that was intended, it is possible that the paths in use will not be disjoint.
In Fig. 5.9 can be seen that three route replies have been sent to the source but only two disjoint paths can be constructed. The backward paths are not removed and constructing path from the routing tables will return three not disjoint paths.

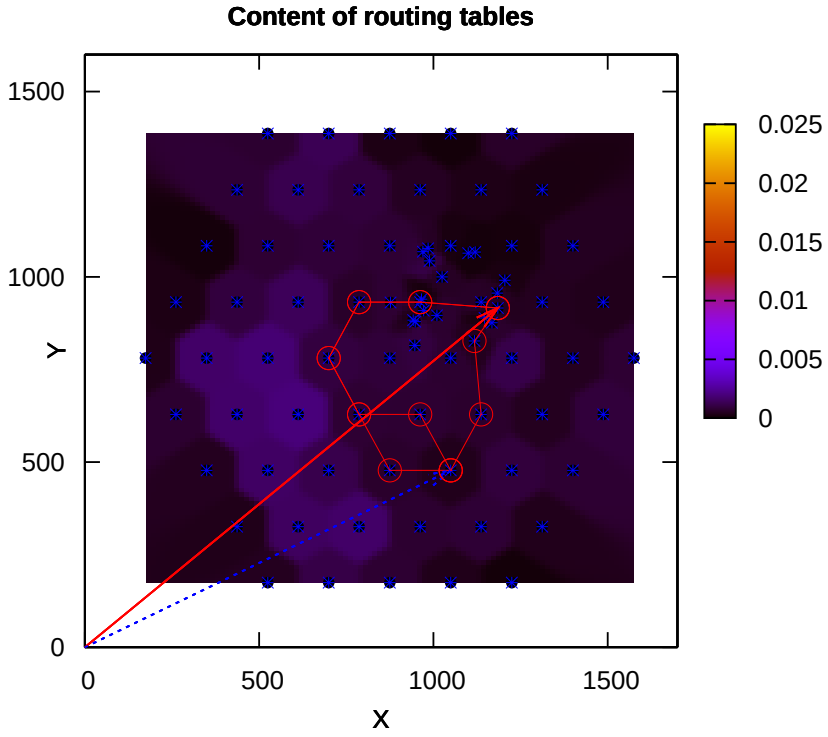

Figure 5.9: Run condition in AOMDV

6. Further Work

Based on the performed investigations and identified problems of AOMDV two directions for further work are proposed. The first is about a new protocol that does not suffer from the same problems as AOMDV, the second direction is about how to improve AOMDV to circumvent these problems.

6.1. AODV-Multipath

One of the main problems of AOMDV is that the flooding algorithm is not designed such that it will find many paths. Another protocol also based on AODV uses a different method of finding multiple paths. This protocol has been proposed in [14] and is called AODVM (AODV-Multipath). If a flooding is initiated intermediate nodes in AODV rebroadcast only one RREQ. In AODVM the duplicates of the RREQ are not discarded but their information are stored. For each RREQ the source, destination and last hop is stored. In contrast to AODV intermediate nodes are not allowed to send a RREP. All the RREP contain the information about the last hop over which the RREQ was received. For each RREQ received at the destination one RREP with the corresponding last hop information is generated and sent to the last hop. Any node receiving a RREP deletes the stored information of the corresponding RREQ and add a routing entry. The RREP is forwarded via the shortest path (that has not yet been used) to the source. Thanks to this enhanced flooding the protocol can find more paths than using the flooding mechanism of AODV. The results in [14] are promissing a quite high number of node-disjoint paths and therefore this protocol could be a candidate if multiple paths to a destination are required.

6.2. Improving AOMDV

One problem with AOMDV is that links are deleted when they seem to be failed. As the protocol can not distinguish between congested and broken links, highly congested paths are removed. If a node is in a congested area, it is possible that all paths will be highly loaded but removing these links and then initiating a new flooding does not solve the problem. The flooding will increase the load and the paths that will be found are still congested. Therefore it could make sense to prevent the protocol from removing congested links. But this requires a method to distinguish between congested and broken links for instance using *Hello* messages. This idea can only solve the problem of loosing paths over time but the problem of not finding enough paths remains.
With the used flooding mechanism only a few paths can be found to a destination. To resolve this problem large changes on the flooding procedure are required. As AODVM already has proposed an enhanced way of flooding, selecting AODVM might be the more promising way than trying to improve AOMDV.

7. Conclusion

7.1. Timeplan

The thesis started with about two week to get familiar with the topic and the simulator. Then the different scheduling algorithms had to be implemented and evaluated. This work was done according to the time plan. As the scheduling did not show great influence on the performance the protocol was further investigated and no time was spent for developping new scheduling methods based on the original problem. As the protocol proved to be not suitable for multipath scheduling the direction of the work was changed and the new goal was to develop tools to show that the protocol is not suitable and perform simulations as a proof. Therefore the second part of the timeplan was not fullfiled and the new topics were planned according to the respective results.

7.2. Protocol

As basis for the thesis the AOMDV protocol should be used. But during the work this protocol showed different problems and the conclusion is that it can not be used for multipath scheduling. The main problem of AOMDV is that it uses a simple flooding mechanism to find paths to the destination. This is well suited for a protocol that uses one path and has the rest as backup. For scheduling over multiple paths, this flooding can not be used, as only some of the existing paths can be found, furthermore the longer the paths to a destination are, the less paths are found. As scheduling makes only sense if the paths do not influence each other (because of interference), short paths are not of interest however it is rare that more than one long path can be found.
The second problem is that the protocol deletes congested paths as they can not be distinguished from broken paths. Deleting paths always increases the chance of sending new route requests and consequentially increase the load of the network. As a consequence, even if routes do not break due to mobility (static scenario) the number of paths from source to destination will decrease over time.
The two above mentioned problems lead to the conclusion that AOMDV can not be used for multipath scheduling.

7.3. Goals Reached

As a start the simulator has been installed and different scheduling methods have been implemented.

- The scheduling function has been designed such that it allows easily to add new scheduling methods by adding a new case to the switch.
- The performance of the methods have been evaluated and compared to the original protocol.
- As the results were not satisfying different evaluation tools have been developed to investigate certain properties of the protocol.
- These tools have been designed such that they can be used with any protocol that can generate the same output.

During the work it was shown that the protocol is not suited for the given problem formulation and therefore the goal was to show the different problems of AOMDV and investigate a special problem. The ques-

tion whether node density is a good scheduling metric in a hotspot scenario has not been answered completely. As any answer is directly related to the underlying protocol, a conclusion would not be generally valid. It was shown that the hotspot is not the area with the highest drop rate but any further conclusion were not possible as the results didn't show a clear trend.
To show the behaviour and the performance of the protocol different tools have been developed. These scripts can produce graphical output and visualise different aspects of the routing protocol.
As the thesis was based on goals that have been defined during the work and the original goals have been omitted, the timeplan was not fullfiled and the original goals were only partly reached. The redefined goals have been reached as different scripts have been developed and the hotspot scenario has been evaluated.

Bibliography

[1] M. Marina and S. Das, "On-demand Multipath Distance Vector Routing in Ad Hoc Networks," in *IEEE International Conference on Network Protocols*, 2001, pp. 14–23.

[2] E. Royer and C.-K. Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks," in *IEEE Personal Communications*, April 1999.

[3] C. Perkins, E. Belding-Royer, and S. Das, "RFC 3561," http://www.ietf.org/rfc/rfc3561.txt, July 2003.

[4] S. Mueller, R. Tsang, and D. Ghosal, "Multipath Routing in Mobile Ad Hoc Networks: Issues and Challenges," in *Invited paper in Lecture Notes in Computer Science*, 2004.

[5] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. Yang, "Load Balancing of Multipath Source Routing in Ad Hoc Networks," in *IEEE ICC 2002*, 2001.

[6] E. Royer, P. Melliar-Smith, and L. Moser, "An Analysis of the Optimum Node Density for Ad Hoc Mobile Networks," in *IEEE International Conference on Communications*, June 2001.

[7] K. Wu and J. Harms, "Load-Sensitive Routing for Mobile Ad Hoc Networks," in *10th IEEE International Conference on Computer Communications and Networks*, October 2001, pp. 540–546.

[8] H. Hassanein and A. Zhou, "Routing with Load Balancing in Wireless Ad Hoc Networks," in *4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2001.

[9] "Network Simulator ns-2," http://www.isi.edu/nsnam/ns/.

[10] M. Marina and S. Das, "NS implementation of AOMDV," http://www.cs.sunysb.edu/∼mahesh/aomdv/.

[11] "Condor," http://computing.ee.ethz.ch/programming/condor.de.html.

[12] "Condor project homepage," http://www.cs.wisc.edu/condor/.

[13] H. Dinsen-Hansen, "Whirlgif," http://www.danbbs.dk/ dino/whirlgif/.

[14] Z. Ye, S. Krishnamurthy, and S. Tripathi, "A framework for Reliable Routing in Mobile Ad Hoc Networks," in *NFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 2003, pp. 270–280.

A. Code

A.1. C++

A.1.1. aodv.h

```
/* -*-  Mode:C++; c-basic-offset:4; tab-width:4; indent-tabs-mode:t -*- */
/*
Copyright (c) 1997, 1998 Carnegie Mellon University. All Rights
Reserved.

Permission to use, copy, modify, and distribute this software and its documentation is
hereby granted (including for commercial or for-profit use), provided that both the
copyright notice and this permission notice appear in all copies of the software,
derivative works, or modified versions, and any portions thereof, and that both notices
appear in supporting documentation, and that credit is given to Carnegie Mellon University
in all publications reporting on direct or indirect use of this code or its derivatives.

ALL CODE, SOFTWARE, PROTOCOLS, AND ARCHITECTURES DEVELOPED BY THE CMU MONARCH PROJECT ARE
EXPERIMENTAL AND ARE KNOWN TO HAVE BUGS, SOME OF WHICH MAY HAVE SERIOUS CONSEQUENCES.
CARNEGIE MELLON PROVIDES THIS SOFTWARE OR OTHER INTELLECTUAL PROPERTY IN ITS ''AS IS''
CONDITION, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE OR INTELLECTUAL PROPERTY, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.

Carnegie Mellon encourages (but does not require) users of this software or intellectual
property to return any improvements or extensions that they make, and to grant Carnegie
Mellon the rights to redistribute these changes without encumbrance.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and
Mahesh Marina, University of Cincinnati. The work was partially done in Sun Microsystems.

#-----------------------------------------------------------------------
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#
# Date:     July 2006
#
# The code has been modiefied by R. Goenner and D.Schatzmann,
# modifications are indicated with the tag // [scheduling] ---
# and they can be activated by using #define SA_CHANGE
#-----------------------------------------------------------------------

*/

#ifndef __aodv_h__
#define __aodv_h__
```

```
#include <iostream>
#include <cmu-trace.h>
#include <priqueue.h>
#include <aodv/aodv_rtable.h>
#include <aodv/aodv_rqueue.h>

#ifdef AOMDV
#define AOMDV_PACKET_SALVAGING
#define AOMDV_MAX_SALVAGE_COUNT      10
#endif // AOMDV

#define AODV_EXPANDING_RING_SEARCH
/*
 Allows local repair of routes
*/
//#define AODV_LOCAL_REPAIR

/*
 Allows AODV to use link-layer (802.11) feedback in determining when
 links are up/down.
*/
#define AODV_LINK_LAYER_DETECTION
#define AODV_HELLO

// [scheduling] ----
// #define AODV_PING
#define AODV_INFO
// #define SA_DEBUG
// [scheduling] ---- end
/*
 Causes AODV to apply a "smoothing" function to the link layer feedback
 that is generated by 802.11. In essence, it requires that RT_MAX_ERROR
 errors occurs within a window of RT_MAX_ERROR_TIME before the link
 is considered bad.
*/
#define AODV_USE_LL_METRIC

/*
 Only applies if AODV_USE_LL_METRIC is defined.
 Causes AODV to apply omniscient knowledge to the feedback received
 from 802.11. This may be flawed, because it does not account for
 congestion.
*/
//#define AODV_USE_GOD_FEEDBACK

class AODV;
#define MY_ROUTE_TIMEOUT     10  // 10 seconds
#define ACTIVE_ROUTE_TIMEOUT 10  // 10 seconds
#define REV_ROUTE_LIFE       6   // 6 seconds
#define BCAST_ID_SAVE        6   // 6 seconds

// [scheduling] ----
// we need a timeout for each path, set it the same as the route timeout
#define ACTIVE_PATH_TIMEOUT 10  // 10 seconds
// [scheduling] ---- end

// No. of times to do network-wide search before timing out for
// MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES         3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT     10.0 // sec

// Should be set by the user using best guess (conservative)
#define NETWORK_DIAMETER     30  // 30 hops

/* Various constants used for the expanding ring search */
#ifdef AODV_EXPANDING_RING_SEARCH
```

```
#define TTL_START            5    // 5
#define TTL_INCREMENT        2    // 2
#else // NO EXPANDING RING SEARCH
#define TTL_START        NETWORK_DIAMETER     // 5
#define TTL_INCREMENT    NETWORK_DIAMETER     // 2
#endif // NO EXPANDING RING SEARCH

#define TTL_THRESHOLD        7

// This should be somewhat related to arp timeout
#define NODE_TRAVERSAL_TIME      0.03     // 30 ms
#define LOCAL_REPAIR_WAIT_TIME   0.15     // sec

// Must be larger than the time difference between a node propagates a route
// request and gets the route reply back.

#define RREP_WAIT_TIME       1.0      // sec

#define ID_NOT_FOUND         0x00
#define ID_FOUND             0x01

// The followings are used for the forward() function. Controls pacing.
#define DELAY                1.0      // random delay
#define NO_DELAY             -1.0     // no delay

// think it should be 30 ms
#define ARP_DELAY            0.01     // fixed delay to keep arp happy

#define HELLO_INTERVAL       1    // 1000 ms
#define ALLOWED_HELLO_LOSS   3    // packets
#define BAD_LINK_LIFETIME    3    // 3000 ms
#define MaxHelloInterval     (1.25 * HELLO_INTERVAL)
#define MinHelloInterval     (0.75 * HELLO_INTERVAL)

// [scheduling] ----
#define PING_INTERVAL        4    // 4s
// [scheduling] ---- end

/*
 Timers (Broadcast ID, Hello, Neighbor Cache, Route Cache)
*/
class BroadcastTimer : public Handler
{
    public:
        BroadcastTimer(AODV* a) : agent(a) {}
        void handle(Event*);
    private:
        AODV *agent;
        Event intr;
};

class HelloTimer : public Handler
{
    public:
        HelloTimer(AODV* a) : agent(a) {}
        void handle(Event*);
    private:
        AODV *agent;
        Event intr;
};

class NeighborTimer : public Handler
{
    public:
        NeighborTimer(AODV* a) : agent(a) {}
        void handle(Event*);
    private:
```

```
        AODV *agent;
        Event intr;
};

class RouteCacheTimer : public Handler
{
    public:
        RouteCacheTimer(AODV* a) : agent(a) {}
        void handle(Event*);
    private:
        AODV *agent;
        Event intr;
};

class LocalRepairTimer : public Handler
{
    public:
        LocalRepairTimer(AODV* a) : agent(a) {}
        void handle(Event*);
    private:
        AODV *agent;
        Event intr;
};

// [scheduling] ——
#ifdef SA_CHANGE
class PingTimer : public Handler
{
public:
    PingTimer(AODV* a) : agent(a) {}
    void handle(Event*);
private:
    AODV *agent;
    Event intr;
};
#endif
// [scheduling] —— end

#ifdef AOMDV
/*
 Route List
*/
class AODV_Route
{
    friend class BroadcastID;
    public:
        AODV_Route(nsaddr_t nexthop, nsaddr_t lasthop=0)
                { nh_addr = nexthop; lh_addr = lasthop;}
    protected:
        LIST_ENTRY(AODV_Route) route_link;
        nsaddr_t nh_addr;
        nsaddr_t lh_addr;
        // [scheduling] ——
        #ifdef SA_CHANGE
        int sched_type;
        #endif
        // [scheduling] —— end
};

LIST_HEAD(aodv_routes, AODV_Route);
#endif // AOMDV


/*
 Broadcast ID Cache
*/
class BroadcastID
```

```
{
    friend class AODV;
    public:
        BroadcastID(nsaddr_t i, u_int32_t b)
        {
            src = i;
            id = b;
            #ifdef AOMDV
            count=0;
            LIST_INIT(&reverse_path_list);
            LIST_INIT(&forward_path_list);
            #endif // AOMDV
        }
    protected:
        LIST_ENTRY(BroadcastID) link;
        nsaddr_t src;
        u_int32_t id;
        double expire;          // now + BCAST_ID_SAVE s

        #ifdef AOMDV
        int count;
        // List of reverse paths used for forwarding replies
        aodv_routes reverse_path_list;
        // List of forward paths advertised already
        aodv_routes forward_path_list;

        inline AODV_Route*
        reverse_path_insert(nsaddr_t nexthop, nsaddr_t lasthop=0)
        {
            AODV_Route* route = new AODV_Route(nexthop, lasthop);

            assert(route);
            LIST_INSERT_HEAD(&reverse_path_list, route, route_link);
            return route;
        }

        inline AODV_Route*
        reverse_path_lookup(nsaddr_t nexthop, nsaddr_t lasthop=0)
        {
            AODV_Route *route = reverse_path_list.lh_first;

            // Search the list for a match of id
            for( ; route; route = route->route_link.le_next)
            {
                if ( (route->nh_addr == nexthop) && (route->lh_addr == lasthop) )
                    return route;
            }
            return NULL;
        }

        inline AODV_Route*
        forward_path_insert(nsaddr_t nexthop, nsaddr_t lasthop=0)
        {
            AODV_Route* route = new AODV_Route(nexthop, lasthop);

            assert(route);
            LIST_INSERT_HEAD(&forward_path_list, route, route_link);
            return route;
        }

        inline AODV_Route*
        forward_path_lookup(nsaddr_t nexthop, nsaddr_t lasthop=0)
        {
            AODV_Route *route = forward_path_list.lh_first;

            // Search the list for a match of id
            for( ; route; route = route->route_link.le_next)
```

```
            {
                if ( (route->nh_addr == nexthop) &&  (route->lh_addr == lasthop) )
                    return route;
            }
            return NULL;
        }
#endif // AOMDV
};

LIST_HEAD(aodv_bcache, BroadcastID);


/*
A struct to store infos about the current node
*/
#ifdef AODV_INFO

struct info {
    nsaddr_t    src;            // src of the path
    nsaddr_t    dst;            // destination of the path
    int         action;         // action (add or remove)
    int         path_count;     // number of path to dest
    nsaddr_t    next_hop;       // next hop of the path
    nsaddr_t    last_hop;       // last hop of the path
    int         hop_count;      // number of hops to the destination
    double      rtt;            // rtt over this path


};

#endif //INFO


/*
 The Routing Agent
*/
class AODV: public Agent
{

    /*
    * make some friends first
    */

    friend class aodv_rt_entry;
    // [scheduling] ---
    #ifdef SA_CHANGE
    friend class AODV_path;
    #endif
    // [scheduling] --- end
    friend class BroadcastTimer;
    friend class HelloTimer;
    friend class NeighborTimer;
    friend class RouteCacheTimer;
    friend class LocalRepairTimer;
    // [scheduling] ---
    #ifdef SA_CHANGE
    #ifdef AODV_PING
    friend class PingTimer;
    #endif
    #ifdef AODV_INFO
    friend class InfoTimer;
    #endif
    #endif
    // [scheduling] --- end
```

```
public:
    AODV(nsaddr_t id);

    void recv(Packet *p, Handler *);

protected:
    int command(int, const char *const *);
    int initialized() { return 1 && target_; }

    /*
    * Route Table Management
    */
    void rt_resolve(Packet *p);

    #ifndef AOMDV
    void rt_update(aodv_rt_entry *rt, u_int32_t seqnum, u_int16_t metric,
                   nsaddr_t nexthop, double expire_time);
    #endif // AOMDV

    void rt_down(aodv_rt_entry *rt);
    void local_rt_repair(aodv_rt_entry *rt, Packet *p);
public:
    void rt_ll_failed(Packet *p);

    #ifndef AOMDV
    void handle_link_failure(nsaddr_t id);
    #else
    void handle_link_failure(nsaddr_t id, bool error=true);
    #endif
protected:
    void rt_purge(void);

    void enque(aodv_rt_entry *rt, Packet *p);
    Packet* deque(aodv_rt_entry *rt);

    /*
    * Neighbor Management
    */
    void nb_insert(nsaddr_t id);
    AODV_Neighbor* nb_lookup(nsaddr_t id);
    void nb_delete(nsaddr_t id);
    void nb_purge(void);

    /*
    * Broadcast ID Management
    */

    BroadcastID* id_insert(nsaddr_t id, u_int32_t bid);
    BroadcastID* id_lookup(nsaddr_t id, u_int32_t bid);
    void id_purge(void);

    /*
    * Packet TX Routines
    */
    void forward(aodv_rt_entry *rt, Packet *p, double delay);
    void forwardReply(aodv_rt_entry *rt, Packet *p, double delay);
    void sendHello(void);
    void sendRequest(nsaddr_t dst);
    // [scheduling] ——
    #ifdef SA_CHANGE
    #ifdef AODV_PING
    void sendPing(void);
    void sendPong(nsaddr_t dest, Packet *p);
    #endif
    #ifdef AODV_INFO
    void sendInfo();
    void dumpPath( aodv_rt_entry *rt, AODV_Path *path, int action, int num_path);
```

```
#endif
#endif
// [scheduling] ---- end

#ifndef AOMDV
void sendReply(nsaddr_t ipdst,
           u_int32_t    hop_count,
           nsaddr_t     rpdst,
           u_int32_t    rpseq,
           u_int32_t    lifetime,
           double       timestamp);
#else // AOMDV
void sendReply(nsaddr_t ipdst,
           u_int32_t    hop_count,
           nsaddr_t     rpdst,
           u_int32_t    rpseq,
           u_int32_t    lifetime,
           double       timestamp,
           nsaddr_t     nexthop,
           u_int32_t    bcast_id,
           nsaddr_t     rp_first_hop);
#endif // AOMDV
void sendError(Packet *p, bool jitter = true);

/*
* Packet RX Routinesaomdv_max_paths_
*/
void recvAODV(Packet *p);
void recvHello(Packet *p);
void recvRequest(Packet *p);
void recvReply(Packet *p);
void recvError(Packet *p);
// [scheduling] ----
#ifdef SA_CHANGE
#ifdef AODV_PING
void recvPing(Packet *p);
#endif
#endif
// [scheduling] ---- end

/*
* History management
*/

double PerHopTime(aodv_rt_entry *rt);


nsaddr_t index;          // IP Address of this node
u_int32_t seqno;         // Sequence Number
int bid;                 // Broadcast ID

aodv_rtable rthead;      // routing table
aodv_ncache nbhead;      // Neighbor Cache
aodv_bcache bihead;      // Broadcast ID Cache
// [scheduling] ----
#ifdef SA_CHANGE
int sched_type;          // scheduler type
#endif
// [scheduling] ---- end

/*
* Timers
*/
BroadcastTimer      btimer;
HelloTimer          htimer;
NeighborTimer       ntimer;
RouteCacheTimer     rtimer;
```

```
        LocalRepairTimer    lrtimer;
        // [scheduling] ——
        #ifdef SA_CHANGE
        PingTimer           ptimer;
        #endif
        // [scheduling] —— end


        /*
        * Routing Table
        */
        aodv_rtable rtable;
        /*
        * A "drop-front" queue used by the routing layer to buffer
        * packets to which it does not have a route.
        */
        aodv_rqueue rqueue;

        /*
        * A mechanism for logging the contents of the routing
        * table.
        */
        Trace *logtarget;

        /*
        * A pointer to the network interface queue that sits
        * between the "classifier" and the "link layer".
        */
        PriQueue *ifqueue;

        /*
        * Logging stuff
        */
        void log_link_del(nsaddr_t dst);
        void log_link_broke(Packet *p);
        void log_link_kept(nsaddr_t dst);

        #ifdef AOMDV
        int aomdv_max_paths_;
        int aomdv_prim_alt_path_len_diff_;
        #endif //AOMDV

        // [scheduling] ——
        #ifdef SA_CHANGE
        #ifdef AODV_INFO
        info node_info; // struct to store informations
        #endif
        #endif
        // [scheduling] —— end
};

#endif /* __aodv_h__ */
```

Listing A.1: aodv.h

A.1.2. aodv.cc

```
/* -*-  Mode:C++; c-basic-offset:4; tab-width:4; indent-tabs-mode:t -*- */
/*
Copyright (c) 1997, 1998 Carnegie Mellon University. All Rights
Reserved.

Permission to use, copy, modify, and distribute this software and its documentation is
hereby granted (including for commercial or for-profit use), provided that both the
copyright notice and this permission notice appear in all copies of the software,
derivative works, or modified versions, and any portions thereof, and that both notices
appear in supporting documentation, and that credit is given to Carnegie Mellon University
in all publications reporting on direct or indirect use of this code or its derivatives.

ALL CODE, SOFTWARE, PROTOCOLS, AND ARCHITECTURES DEVELOPED BY THE CMU MONARCH PROJECT ARE
EXPERIMENTAL AND ARE KNOWN TO HAVE BUGS, SOME OF WHICH MAY HAVE SERIOUS CONSEQUENCES.
CARNEGIE MELLON PROVIDES THIS SOFTWARE OR OTHER INTELLECTUAL PROPERTY IN ITS ''AS IS''
CONDITION, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OF THIS SOFTWARE OR INTELLECTUAL PROPERTY, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

Carnegie Mellon encourages (but does not require) users of this software or intellectual
property to return any improvements or extensions that they make, and to grant Carnegie
Mellon the rights to redistribute these changes without encumbrance.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and
Mahesh Marina, University of Cincinnati. The work was partially done in Sun Microsystems.
Modified for gratuitous replies by Anant Utgikar, 09/16/02.

#-----------------------------------------------------------------------
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#
# Date:     July 2006
#
# The code has been modiefied by R. Goenner and D.Schatzmann,
# modifications are indicated with the tag // [scheduling] ---
# and they can be activated by using #define SA_CHANGE
#-----------------------------------------------------------------------

*/

//#include <ip.h>

#include <aodv/aodv.h>
#include <aodv/aodv_packet.h>
#include <random.h>
#include <cmu-trace.h>
#include <iostream>


#define max(a,b) ( (a) > (b) ? (a) : (b) )
#define NOW Scheduler::instance().clock()

//#define DEBUG
//#define ERROR
```

```
#ifdef DEBUG
static int extra_route_reply = 0;
static int limit_route_request = 0;
static int route_request = 0;
#endif


/*
 TCL Hooks
*/


int hdr_aodv::offset_;
static class AODVHeaderClass : public PacketHeaderClass
{
    public:
    AODVHeaderClass() : PacketHeaderClass("PacketHeader/AODV", sizeof(hdr_all_aodv))
    {
        bind_offset(&hdr_aodv::offset_);
    }
} class_rtProtoAODV_hdr;

static class AODVclass : public TclClass
{
    public:
    AODVclass() : TclClass("Agent/AODV") {}
    TclObject* create(int argc, const char*const* argv)
    {
        assert(argc == 5);
        return (new AODV((nsaddr_t) Address::instance().str2addr(argv[4])));
    }
} class_rtProtoAODV;


int AODV::command(int argc, const char*const* argv)
{
    if(argc == 2)
    {
        Tcl& tcl = Tcl::instance();

        if(strncasecmp(argv[1], "id", 2) == 0)
        {
            tcl.resultf("%d", index);
            return TCL_OK;
        }
        if (strncasecmp(argv[1], "dump-table", 10) == 0)
        {
            printf("Node %d: Route table:\n", index);
            rtable.rt_dumptable();
            return TCL_OK;
        }
        if(strncasecmp(argv[1], "start", 2) == 0)
        {
            btimer.handle((Event*) 0);

            #ifndef AODV_LINK_LAYER_DETECTION
            htimer.handle((Event*) 0);
            ntimer.handle((Event*) 0);
            #endif // LINK LAYER DETECTION


            #ifdef AODV_HELLO
            htimer.handle((Event*) 0);
            ntimer.handle((Event*) 0);
            #endif // AODV HELLO
            rtimer.handle((Event*) 0);
            // [scheduling] ——
```

```
            #ifdef SA_CHANGE
            #ifdef AODV_PING
            ptimer.handle((Event*) 0);
            #endif
            #endif
            // [scheduling] ---- end
            return TCL_OK;
        }
    }

    else if(argc == 3)
    {
            if(strcmp(argv[1], "index") == 0)
            {
            index = atoi(argv[2]);
            return TCL_OK;
        }

        else if(strcmp(argv[1], "log-target") == 0 ||
            strcmp(argv[1], "tracetarget") == 0)
        {
            logtarget = (Trace*) TclObject::lookup(argv[2]);
            if(logtarget == 0)
            return TCL_ERROR;
            return TCL_OK;
        }
        else if(strcmp(argv[1], "drop-target") == 0)
        {
        int stat = rqueue.command(argc,argv);
            if (stat != TCL_OK) return stat;
            return Agent::command(argc, argv);
        }
        else if(strcmp(argv[1], "if-queue") == 0)
        {
            ifqueue = (PriQueue*) TclObject::lookup(argv[2]);

            if(ifqueue == 0)
            return TCL_ERROR;
            return TCL_OK;
        }
    }
    return Agent::command(argc, argv);
}

/*
 Constructor
*/
#ifdef SA_CHANGE
AODV::AODV(nsaddr_t id) : Agent(PT_AODV), btimer(this), htimer(this),
    ntimer(this), rtimer(this), lrtimer(this),ptimer(this), rqueue()
#else
AODV::AODV(nsaddr_t id) : Agent(PT_AODV), btimer(this), htimer(this),
    ntimer(this), rtimer(this), lrtimer(this), rqueue()
#endif
{
    #ifdef AOMDV
    //aomdv_max_paths_ = 2;
    bind("aomdv_max_paths_", &aomdv_max_paths_);
    // [scheduling] ----
    #ifdef SA_CHANGE
    rtable.aomdv_max_paths_= aomdv_max_paths_;      // set it in the rtable
    aomdv_prim_alt_path_len_diff_ = 15;             // original setting was 5
    #else
    // [scheduling] ----  end
    rtable.aomdv_max_paths_= aomdv_max_paths_;      // set it in the rtable
    aomdv_prim_alt_path_len_diff_ = 5;
    #endif
```

```
    #endif //AOMDV

    index = id;
    seqno = 2;
    bid = 1;

    // [scheduling] ----
    #ifdef SA_CHANGE
    bind("sched_type", &sched_type);
    rtable.sched_type = sched_type;
    #endif
    // [scheduling] ----  end
    LIST_INIT(&nbhead);
    LIST_INIT(&bihead);

    logtarget = 0;
    ifqueue = 0;
}

/*
 Timers
*/

void BroadcastTimer::handle(Event*)
{
    agent->id_purge();
    Scheduler::instance().schedule(this, &intr, BCAST_ID_SAVE);
}


void HelloTimer::handle(Event*)
{
    /* Do not send a HELLO message unless we have a valid route entry. */
    if (agent->rtable.rt_has_active_route())
        agent->sendHello();
    // CHANGE
    double interval = HELLO_INTERVAL + 0.01*Random::uniform();
    assert(interval >= 0);
    Scheduler::instance().schedule(this, &intr, interval);
}

void NeighborTimer::handle(Event*)
{
    agent->nb_purge();
    Scheduler::instance().schedule(this, &intr, HELLO_INTERVAL);
}


void RouteCacheTimer::handle(Event*)
{
    agent->rt_purge();
    #define FREQUENCY 0.5 // sec
    Scheduler::instance().schedule(this, &intr, FREQUENCY);
}

void LocalRepairTimer::handle(Event* p)  // SRD: 5/4/99
{
    aodv_rt_entry *rt;
    struct hdr_ip *ih = HDR_IP( (Packet *)p);

    /* you get here after the timeout in a local repair attempt */
    /* fprintf(stderr, "%s\n", __FUNCTION__); */


    rt = agent->rtable.rt_lookup(ih->daddr());

    if (rt && rt->rt_flags != RTF_UP)
```

```
    {
        // route is yet to be repaired
        // I will be conservative and bring down the route
        // and send route errors upstream.
        /* The following assert fails, not sure why */
        /* assert (rt->rt_flags == RTF_IN_REPAIR); */

        agent->rt_down(rt);
        // send RERR
        #ifdef DEBUG
        fprintf(stderr,"Node %d: Dst - %d, failed local repair\n", index, rt->rt_dst);
        #endif
    }
    Packet::free((Packet *)p);
}

// [scheduling] ---
#ifdef SA_CHANGE
void PingTimer::handle(Event*)
{
    #ifdef AODV_PING
    agent->sendPing();

    double interval = PING_INTERVAL + 0.01*Random::uniform();
    assert(interval >= 0);
    Scheduler::instance().schedule(this, &intr, interval);
    #endif
}
#endif

// [scheduling] --- end


/*
 Broadcast ID Management Functions
*/


BroadcastID* AODV::id_insert(nsaddr_t id, u_int32_t bid)
{
    BroadcastID *b = new BroadcastID(id, bid);

    assert(b);
    b->expire = NOW + BCAST_ID_SAVE;
    LIST_INSERT_HEAD(&bihead, b, link);
    return b;
}

/* SRD */
BroadcastID* AODV::id_lookup(nsaddr_t id, u_int32_t bid)
{
    BroadcastID *b = bihead.lh_first;

    // Search the list for a match of source and bid
    for( ; b; b = b->link.le_next)
    {
        if ((b->src == id) && (b->id == bid))
        return b;
    }
    return NULL;
}

void AODV::id_purge()
{
    BroadcastID *b = bihead.lh_first;
    BroadcastID *bn;
    double now = NOW;
```

```
    for(; b; b = bn)
    {
        bn = b->link.le_next;
        if(b->expire <= now)
        {
            LIST_REMOVE(b,link);
            delete b;
        }
    }
}

/*
 Helper Functions
*/

double AODV::PerHopTime(aodv_rt_entry *rt)
{
    int num_non_zero = 0, i;
    double total_latency = 0.0;

    if (!rt)
    return ((double) NODE_TRAVERSAL_TIME );

    for (i=0; i < MAX_HISTORY; i++)
    {
        if (rt->rt_disc_latency[i] > 0.0)
        {
            num_non_zero++;
            total_latency += rt->rt_disc_latency[i];
        }
    }
    if (num_non_zero > 0)
        return(total_latency / (double) num_non_zero);
    else
        return((double) NODE_TRAVERSAL_TIME);
}

/*
 Link Failure Management Functions
*/

static void aodv_rt_failed_callback(Packet *p, void *arg)
{
    ((AODV*) arg)->rt_ll_failed(p);
}

/*
 * This routine is invoked when the link-layer reports a route failed.
 */
void AODV::rt_ll_failed(Packet *p)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    aodv_rt_entry *rt;
    nsaddr_t broken_nbr = ch->next_hop_;

    #ifndef AODV_LINK_LAYER_DETECTION
    drop(p, DROP_RTR_MAC_CALLBACK);
    #else

    /*
    * Non-data packets and Broadcast Packets can be dropped.
    */
    if(! DATA_PACKET(ch->ptype()) || (u_int32_t) ih->daddr() == IP_BROADCAST)
    {
        drop(p, DROP_RTR_MAC_CALLBACK);
```

```
    return;
}
log_link_broke(p);
if((rt = rtable.rt_lookup(ih->daddr())) == 0)
{
    drop(p, DROP_RTR_MAC_CALLBACK);
    return;
}
log_link_del(ch->next_hop_);

#ifdef AODV_LOCAL_REPAIR
/* if the broken link is closer to the dest than source,
attempt a local repair. Otherwise, bring down the route. */


if (ch->num_forwards() > rt->rt_hops)
{
    local_rt_repair(rt, p); // local repair
    // retrieve all the packets in the ifq using this link,
    // queue the packets for which local repair is done,
    return;
}
#endif // LOCAL REPAIR

handle_link_failure(broken_nbr);

#ifdef AOMDV_PACKET_SALVAGING

if ( !DATA_PACKET(ch->ptype()) )
    drop(p, DROP_RTR_MAC_CALLBACK);
else
{
    // salvage the packet using an alternate path if available.
    aodv_rt_entry *rt = rtable.rt_lookup(ih->daddr());
    if ( rt &&
        (rt->rt_flags == RTF_UP) &&
        (ch->aomdv_salvage_count_ < AOMDV_MAX_SALVAGE_COUNT) )
    {
        ch->aomdv_salvage_count_ += 1;
        forward(rt, p, NO_DELAY);
    }
    else drop(p, DROP_RTR_MAC_CALLBACK);
}
while((p = ifqueue->filter(broken_nbr)))
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    if ( !DATA_PACKET(ch->ptype()) )
        drop(p, DROP_RTR_MAC_CALLBACK);
    else
    {
        // salvage the packet using an alternate path if available.
        aodv_rt_entry *rt = rtable.rt_lookup(ih->daddr());
        if ( rt &&
            (rt->rt_flags == RTF_UP) &&
            (ch->aomdv_salvage_count_ < AOMDV_MAX_SALVAGE_COUNT) )
        {
            ch->aomdv_salvage_count_ += 1;
            forward(rt, p, NO_DELAY);
        }
        else drop(p, DROP_RTR_MAC_CALLBACK);
    }
}

#else // NO PACKET SALVAGING

drop(p, DROP_RTR_MAC_CALLBACK);
```

```
    // Do the same thing for other packets in the interface queue using the
    // broken link -Mahesh
    while((p = ifqueue->filter(broken_nbr)))
    {
        drop(p, DROP_RTR_MAC_CALLBACK);
    }
    nb_delete(broken_nbr);          /* MAYBE THIS SHOULD NOT BE?? */
    #endif // NO PACKET SALVAGING


    #endif // LINK LAYER DETECTION
}

#ifndef AOMDV
void AODV::handle_link_failure(nsaddr_t id)
{
    aodv_rt_entry *rt, *rtn;
    Packet *rerr = Packet::alloc();
    struct hdr_aodv_error *re = HDR_AODV_ERROR(rerr);

    re->DestCount = 0;
    for(rt = rtable.head(); rt; rt = rtn)  // for each rt entry
    {
        rtn = rt->rt_link.le_next;
        if ((rt->rt_hops != INFINITY2) && (rt->rt_nexthop == id) )
        {
            assert (rt->rt_flags == RTF_UP);
            assert((rt->rt_seqno % 2) == 0);
            rt->rt_seqno++;
            re->unreachable_dst[re->DestCount] = rt->rt_dst;
            re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
            #ifdef DEBUG
            fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\n", __FUNCTION__, NOW,
                index, re->unreachable_dst[re->DestCount],
                re->unreachable_dst_seqno[re->DestCount], rt->rt_nexthop);
            #endif // DEBUG
            re->DestCount += 1;
            rt_down(rt);
        }
        // remove the lost neighbor from all the precursor lists
        rt->pc_delete(id);
    }

    if (re->DestCount > 0)
    {
        #ifdef DEBUG
        fprintf(stderr, "%s(%f): %d\tsending RERR...\n", __FUNCTION__, NOW, index);
        #endif // DEBUG
        sendError(rerr, false);
    }
    else
    {
        Packet::free(rerr);
    }
}

void AODV::rt_down(aodv_rt_entry *rt)
{
    /*
    * Make sure that you don't "down" a route more than once.
    */

    if(rt->rt_flags == RTF_DOWN)
    {
        return;
    }
```

```
    rt->rt_last_hop_count = rt->rt_hops;
    rt->rt_hops = INFINITY2;
    rt->rt_flags = RTF_DOWN;
    rt->rt_nexthop = 0;
    rt->rt_expire = 0;

} /* rt_down function */

#endif //AOMDV

void AODV::local_rt_repair(aodv_rt_entry *rt, Packet *p)
{
    #ifdef DEBUG
    fprintf(stderr,"%s: Dst - %d\n", __FUNCTION__, rt->rt_dst);
    #endif
    // Buffer the packet
    rqueue.enque(p);

    // mark the route as under repair
    rt->rt_flags = RTF_IN_REPAIR;

    sendRequest(rt->rt_dst);

    // set up a timer interrupt
    Scheduler::instance().schedule(&lrtimer, p->copy(), rt->rt_req_timeout);
}

/*
 Route Handling Functions
*/

void AODV::rt_resolve(Packet *p)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    aodv_rt_entry *rt;

    /*
    * Set the transmit failure callback. That
    * won't change.
    */
    ch->xmit_failure_ = aodv_rt_failed_callback;
    ch->xmit_failure_data_ = (void*) this;
    rt = rtable.rt_lookup(ih->daddr());
    if(rt == 0)
    {
        rt = rtable.rt_add(ih->daddr());
    }

    /*
    * If the route is up, forward the packet
    */

    if(rt->rt_flags == RTF_UP)
    {
        assert(rt->rt_hops != INFINITY2);
        forward(rt, p, NO_DELAY);
    }
    /*
    * if I am the source of the packet, then do a Route Request.
    */
    else if(ih->saddr() == index)
    {
        rqueue.enque(p);
        sendRequest(rt->rt_dst);
    }
    /*
```

```
    * A local repair is in progress. Buffer the packet.
    */
    else if (rt->rt_flags == RTF_IN_REPAIR)
    {
        rqueue.enque(p);
    }

    /*
    * I am trying to forward a packet for someone else to which
    * I don't have a route.
    */
    else
    {
        Packet *rerr = Packet::alloc();
        struct hdr_aodv_error *re = HDR_AODV_ERROR(rerr);
        /*
        * For now, drop the packet and send error upstream.
        * Now the route errors are broadcast to upstream
        * neighbors - Mahesh 09/11/99
        */

        assert (rt->rt_flags == RTF_DOWN);
        re->DestCount = 0;
        re->unreachable_dst[re->DestCount] = rt->rt_dst;
        re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
        re->DestCount += 1;
        #ifdef DEBUG
        fprintf(stderr, "%s: sending RERR...\n", __FUNCTION__);
        #endif
        sendError(rerr, false);

        drop(p, DROP_RTR_NO_ROUTE);
    }

}

void AODV::rt_purge()
{
    aodv_rt_entry *rt, *rtn;
    double now = NOW;
    double delay = 0.0;
    Packet *p;

    for(rt = rtable.head(); rt; rt = rtn)  // for each rt entry
    {
        rtn = rt->rt_link.le_next;

        #ifndef AOMDV

        if ((rt->rt_flags == RTF_UP) && (rt->rt_expire < now))
        {
            // if a valid route has expired, purge all packets from
            // send buffer and invalidate the route.
            assert(rt->rt_hops != INFINITY2);
            while((p = rqueue.deque(rt->rt_dst)))
            {
                #ifdef DEBUG
                fprintf(stderr, "%s: calling drop()\n", __FUNCTION__);
                #endif // DEBUG
                drop(p, DROP_RTR_NO_ROUTE);
            }
            rt->rt_seqno++;
            assert (rt->rt_seqno % 2);
            rt_down(rt);
        }

        #else // AOMDV
```

```
// [scheduling] ----
#ifndef SA_CHANGE
if (rt->rt_flags == RTF_UP && (rt->rt_expire < now))
#else
/* if (rt->rt_flags == RTF_UP && (rt->rt_expire < now))
 * we want to remove all old paths, even if the rt entry is not expired! */

if (rt->rt_flags == RTF_UP )
#endif
// [scheduling] ---- end
{
    // [scheduling] ----
    #ifdef SA_CHANGE
    /* stupid implementation, but easier like this */
    #ifdef AODV_INFO
    AODV_Path *path = rt->rt_path_list.lh_first;
    int num_path = rt->rt_num_paths_;
    for(; path; path = path->path_link.le_next)
    {
        if(path->expire < now)
        {
            /* print all the path that will be removed*/
            num_path --;
            dumpPath(rt, path, -1, num_path);
        }
    }
    #endif
    #endif
    // [scheduling] ---- end
    rt->path_purge();

    if (rt->path_empty())
    {
        while((p = rqueue.deque(rt->rt_dst)))
        {
            drop(p, DROP_RTR_RTEXPIRE);
        }
        rt->rt_seqno++;
        rt->rt_seqno = max(rt->rt_seqno, rt->rt_highest_seqno_heard);
        if (rt->rt_seqno % 2 == 0)
            rt->rt_seqno += 1;
        rt_down(rt);
    }
}

#endif // AOMDV


if (rt->rt_flags == RTF_UP)
{
    // If the route is not expired,
    // and there are packets in the sendbuffer waiting,
    // forward them. This should not be needed, but this extra
    // check does no harm.
    assert(rt->rt_hops != INFINITY2);
    while((p = rqueue.deque(rt->rt_dst)))
    {
        forward (rt, p, delay);
        delay += ARP_DELAY;
    }
}
else if (rqueue.find(rt->rt_dst))
    // If the route is down and
    // if there is a packet for this destination waiting in
    // the sendbuffer, then send out route request. sendRequest
    // will check whether it is time to really send out request
```

```
            // or not.
            // This may not be crucial to do it here, as each generated
            // packet will do a sendRequest anyway.
            sendRequest(rt->rt_dst);
    }
}

/*
 Packet Reception Routines
*/

void AODV::recv(Packet *p, Handler*)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);

    assert(initialized());
    //assert(p->incoming == 0);
    // XXXXX NOTE: use of incoming flag has been depracated; In order to track
    // direction of pkt flow,
    // direction_ in hdr_cmn is used instead. see packet.h for details.

    if(ch->ptype() == PT_AODV)
    {
        ih->ttl_ -= 1;
        recvAODV(p);
        return;
    }

    /*
    * Must be a packet I'm originating...
    */
    if((ih->saddr() == index) && (ch->num_forwards() == 0))
    {
        /*
        * Add the IP Header
        */
        ch->size() += IP_HDR_LEN;
        #ifdef AOMDV
        ch->aomdv_salvage_count_ = 0;
        #endif // AOMDV
        // Added by Parag Dadhania && John Novatnack to handle broadcasting
        if ( (u_int32_t)ih->daddr() != IP_BROADCAST)
            ih->ttl_ = NETWORK_DIAMETER;
    }
    /*
    * I received a packet that I sent. Probably
    * a routing loop.
    */
    else if(ih->saddr() == index)
    {
        drop(p, DROP_RTR_ROUTE_LOOP);
        return;
    }
    /*
    * Packet I'm forwarding...
    */
    else
    {
        /*
        * Check the TTL. If it is zero, then discard.
        */
        if(--ih->ttl_ == 0)
        {
            drop(p, DROP_RTR_TTL);
            return;
        }
```

```
    }
    // Added by Parag Dadhania && John Novatnack to handle broadcasting
    if ( (u_int32_t)ih->daddr() != IP_BROADCAST)
        rt_resolve(p);
    else
        forward((aodv_rt_entry*) o, p, NO_DELAY);
}


void AODV::recvAODV(Packet *p)
{
    struct hdr_aodv *ah = HDR_AODV(p);
    struct hdr_ip *ih = HDR_IP(p);

    assert(ih->sport() == RT_PORT);
    assert(ih->dport() == RT_PORT);

    /*
    * Incoming Packets.
    */
    switch(ah->ah_type)
    {

        case AODVTYPE_RREQ:
        recvRequest(p);
        break;

        case AODVTYPE_RREP:
        recvReply(p);
        break;

        case AODVTYPE_RERR:
        recvError(p);
        break;

        case AODVTYPE_HELLO:
        recvHello(p);
        break;

        // [scheduling] ----
        #ifdef SA_CHANGE
        #ifdef AODV_PING
        case AODVTYPE_PING:
        recvPing(p);
        break;
        #endif

        #ifdef AODV_INFO
        case AODVTYPE_INFO:
        Packet::free(p); // not needed
        break;
        #endif
        #endif
        // [scheduling] ---- end

        default:
        fprintf(stderr, "Invalid AODV type (%x)\n", ah->ah_type);
        exit(1);
    }

}

#ifndef AOMDV
void AODV::recvRequest(Packet *p)
{
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
```

```
aodv_rt_entry *rt;

/*
* Drop if:
* - I'm the source
* - I recently heard this request.
*/

if(rq->rq_src == index)
{
    #ifdef DEBUG
    fprintf(stderr, "%s: got my own REQUEST\n", __FUNCTION__);
    #endif // DEBUG
    Packet::free(p);
    return;
}

if (id_lookup(rq->rq_src, rq->rq_bcast_id))
{

    #ifdef DEBUG
    fprintf(stderr, "%s: discarding request\n", __FUNCTION__);
    #endif // DEBUG

    Packet::free(p);
    return;
}

/*
* Cache the broadcast ID
*/
id_insert(rq->rq_src, rq->rq_bcast_id);



/*
* We are either going to forward the REQUEST or generate a
* REPLY. Before we do anything, we make sure that the REVERSE
* route is in the route table.
*/
aodv_rt_entry *rto; // rto is the reverse route

rto = rtable.rt_lookup(rq->rq_src);
if(rto == 0)  /* if not in the route table */
{
    // create an entry for the reverse route.
    rto = rtable.rt_add(rq->rq_src);
}

rto->rt_expire = max(rto->rt_expire, (NOW + REV_ROUTE_LIFE));


if ( (rq->rq_src_seqno > rto->rt_seqno ) ||
((rq->rq_src_seqno == rto->rt_seqno) &&
(rq->rq_hop_count < rto->rt_hops)) )
{
    // If we have a fresher seq no. or lesser #hops for the
    // same seq no., update the rt entry. Else don't bother.
    // CHANGE (make sure we increment hop count before adding reverse path
    // route entry.

    rt_update(rto, rq->rq_src_seqno, rq->rq_hop_count + 1, ih->saddr(),
    max(rto->rt_expire, (NOW + REV_ROUTE_LIFE)) );
    if (rto->rt_req_timeout > 0.0)
    {
        // Reset the soft state and
        // Set expiry time to NOW + ACTIVE_ROUTE_TIMEOUT
```

```
            // This is because route is used in the forward direction,
            // but only sources get benefited by this change
            rto->rt_req_cnt = 0;
            rto->rt_req_timeout = 0.0;
            rto->rt_req_last_ttl = rq->rq_hop_count;
            rto->rt_expire = NOW + ACTIVE_ROUTE_TIMEOUT;
        }

        /* Find out whether any buffered packet can benefit from the
        * reverse route.
        * May need some change in the following code - Mahesh 09/11/99
        */
        assert (rto->rt_flags == RTF_UP);
        Packet *buffered_pkt;
        while ((buffered_pkt = rqueue.deque(rto->rt_dst)))
        {
            if (rto && (rto->rt_flags == RTF_UP))
            {
                assert(rto->rt_hops != INFINITY2);
                forward(rto, buffered_pkt, NO_DELAY);
            }
        }
    }
    // End for putting reverse route in rt table


    /*
    * We have taken care of the reverse route stuff.
    * Now see whether we can send a route reply.
    */

    rt = rtable.rt_lookup(rq->rq_dst);

    // First check if I am the destination ..

    if(rq->rq_dst == index)
    {

        #ifdef DEBUG
        fprintf(stderr, "%d - %s: destination sending reply\n", index, __FUNCTION__);
        #endif // DEBUG


        // Just to be safe, I use the max. Somebody may have
        // incremented the dst seqno.
        seqno = max(seqno, rq->rq_dst_seqno)+1;
        if (seqno % 2) seqno++;

        sendReply(  rq->rq_src,         // IP Destination
                1,                      // Hop Count
                index,                  // Dest IP Address
                seqno,                  // Dest Sequence Num
                MY_ROUTE_TIMEOUT,       // Lifetime
                rq->rq_timestamp);      // timestamp

        Packet::free(p);
    }

    // I am not the destination, but I may have a fresh enough route.

    else if (rt && (rt->rt_hops != INFINITY2) &&
        (rt->rt_seqno >= rq->rq_dst_seqno) )
    {

        //assert (rt->rt_flags == RTF_UP);
        assert(rq->rq_dst == rt->rt_dst);
        //assert ((rt->rt_seqno % 2) == 0); // is the seqno even?
```

```
        sendReply(  rq->rq_src,
                rt->rt_hops + 1,
                rq->rq_dst,
                rt->rt_seqno,
                (u_int32_t) (rt->rt_expire - NOW),
                rq->rq_timestamp);
        // Insert nexthops to RREQ source and RREQ destination in the
        // precursor lists of destination and source respectively
        rt->pc_insert(rto->rt_nexthop); // nexthop to RREQ source
        rto->pc_insert(rt->rt_nexthop); // nexthop to RREQ destination

        #ifdef RREQ_GRAT_RREP

        sendReply(  rq->rq_dst,
                rq->rq_hop_count,
                rq->rq_src,
                rq->rq_src_seqno,
                (u_int32_t) (rt->rt_expire - NOW),
                rq->rq_timestamp);
        #endif

        // TODO: send grat RREP to dst if G flag set in RREQ
        // using rq->rq_src_seqno, rq->rq_hop_counT

        // DONE: Included gratuitous replies to be sent as per IETF aodv draft
        // specification. As of now, G flag has not been dynamically used
        // and is always set or reset in aodv-packet.h, Anant Utgikar, 09/16/02.

        Packet::free(p);
    }
    /*
    * Can't reply. So forward the Route Request
    */
    else
    {
        ih->saddr() = index;
        ih->daddr() = IP_BROADCAST;
        rq->rq_hop_count += 1;
        // Maximum sequence number seen en route
        if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);
            forward((aodv_rt_entry*) 0, p, DELAY);
    }

}

void AODV::sendReply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst,
 u_int32_t rpseq, u_int32_t lifetime, double timestamp)
{
    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
    aodv_rt_entry *rt = rtable.rt_lookup(ipdst);

    #ifdef DEBUG
    fprintf(stderr, "sending Reply from %d at %.2f\n", index, NOW);
    #endif // DEBUG
    assert(rt);

    rp->rp_type = AODVTYPE_RREP;
    rp->rp_hop_count = hop_count;
    rp->rp_dst = rpdst;
    rp->rp_dst_seqno = rpseq;
    rp->rp_src = index;
    rp->rp_lifetime = lifetime;
    rp->rp_timestamp = timestamp;
```

```
    ch->ptype() = PT_AODV;
    ch->size() = IP_HDR_LEN + rp->size();
    ch->iface() = -2;
    ch->error() = 0;
    ch->addr_type() = NS_AF_INET;
    ch->next_hop_ = rt->rt_nexthop;
    ch->prev_hop_ = index; // AODV hack
    ch->direction() = hdr_cmn::DOWN;

    ih->saddr() = index;
    ih->daddr() = ipdst;
    ih->sport() = RT_PORT;
    ih->dport() = RT_PORT;
    ih->ttl_ = NETWORK_DIAMETER;

    Scheduler::instance().schedule(target_, p, 0.);

}

void AODV::recvReply(Packet *p)
{
    //struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
    aodv_rt_entry *rt;
    char suppress_reply = 0;
    double delay = 0.0;

    #ifdef DEBUG
    fprintf(stderr, "%d - %s: received a REPLY\n", index, __FUNCTION__);
    #endif // DEBUG


    /*
    * Got a reply. So reset the "soft state" maintained for
    * route requests in the request table. We don't really have
    * have a separate request table. It is just a part of the
    * routing table itself.
    */
    // Note that rp_dst is the dest of the data packets, not the
    // the dest of the reply, which is the src of the data packets.

    rt = rtable.rt_lookup(rp->rp_dst);

    /*
    * If I don't have a rt entry to this host... adding
    */
    if(rt == 0)
    {
        rt = rtable.rt_add(rp->rp_dst);
    }

    /*
    * Add a forward route table entry... here I am following
    * Perkins-Royer AODV paper almost literally - SRD 5/99
    */

    if ( (rt->rt_seqno < rp->rp_dst_seqno) ||   // newer route
         ((rt->rt_seqno == rp->rp_dst_seqno) &&
          (rt->rt_hops > rp->rp_hop_count)) )    // shorter or better route
    {
        // Update the rt entry
        rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count,
        rp->rp_src, NOW + rp->rp_lifetime);

        // reset the soft state
        rt->rt_req_cnt = 0;
```

```
    rt->rt_req_timeout = 0.0;
    rt->rt_req_last_ttl = rp->rp_hop_count;

    if (ih->daddr() == index)       // If I am the original source
    {
        // Update the route discovery latency statistics
        // rp->rp_timestamp is the time of request origination

        rt->rt_disc_latency[rt->hist_indx] = (NOW - rp->rp_timestamp)/
                            (double) rp->rp_hop_count;
        // increment indx for next time
        rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
    }

    /*
    * Send all packets queued in the sendbuffer destined for
    * this destination.
    * XXX - observe the "second" use of p.
    */
    Packet *buf_pkt;
    while((buf_pkt = rqueue.deque(rt->rt_dst)))
    {
        if(rt->rt_hops != INFINITY2)
        {
            assert (rt->rt_flags == RTF_UP);
            // Delay them a little to help ARP. Otherwise ARP
            // may drop packets. -SRD 5/23/99
            forward(rt, buf_pkt, delay);
            delay += ARP_DELAY;
        }
    }
}
else
{
    suppress_reply = 1;
}

/*
* If reply is for me, discard it.
*/

if(ih->daddr() == index || suppress_reply)
{
    Packet::free(p);
}
/*
* Otherwise, forward the Route Reply.
*/
else
{
    // Find the rt entry
    aodv_rt_entry *rto = rtable.rt_lookup(ih->daddr());
    // If the rt is up, forward
    if(rto && (rto->rt_hops != INFINITY2))
    {
        assert (rto->rt_flags == RTF_UP);
        rp->rp_hop_count += 1;
        rp->rp_src = index;
        forward(rto, p, NO_DELAY);
        // Insert the nexthop towards the RREQ source to
        // the precursor list of the RREQ destination
        rt->pc_insert(rto->rt_nexthop); // nexthop to RREQ source

    }
    else
    {
        // I don't know how to forward .. drop the reply.
```

```
            #ifdef DEBUG
            fprintf(stderr, "%s: dropping Route Reply\n", __FUNCTION__);
            #endif // DEBUG
            drop(p, DROP_RTR_NO_ROUTE);
        }
    }

}

void AODV::rt_update(aodv_rt_entry *rt, u_int32_t seqnum, u_int16_t metric,
 nsaddr_t nexthop, double expire_time)
{

    rt->rt_seqno = seqnum;
    rt->rt_hops = metric;
    rt->rt_flags = RTF_UP;
    rt->rt_nexthop = nexthop;
    rt->rt_expire = expire_time;
}
#endif //AOMDV

#ifndef AOMDV
void AODV::recvError(Packet *p)
{
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_error *re = HDR_AODV_ERROR(p);
    aodv_rt_entry *rt;
    u_int8_t i;
    Packet *rerr = Packet::alloc();
    struct hdr_aodv_error *nre = HDR_AODV_ERROR(rerr);

    nre->DestCount = 0;

    for (i=0; i<re->DestCount; i++)
    {
        // For each unreachable destination
        rt = rtable.rt_lookup(re->unreachable_dst[i]);
        if ( rt && (rt->rt_hops != INFINITY2) &&
        (rt->rt_nexthop == ih->saddr()) &&
        (rt->rt_seqno <= re->unreachable_dst_seqno[i]) )
        {
            assert(rt->rt_flags == RTF_UP);
            assert((rt->rt_seqno % 2) == 0); // is the seqno even?
            #ifdef DEBUG
            fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\t(%d\t%u\t%d)\n", __FUNCTION__,NOW,
                    index, rt->rt_dst, rt->rt_seqno, rt->rt_nexthop,
                    re->unreachable_dst[i], re->unreachable_dst_seqno[i], ih->saddr());
            #endif // DEBUG
            rt->rt_seqno = re->unreachable_dst_seqno[i];
            rt_down(rt);

            // Not sure whether this is the right thing to do
            Packet *pkt;
            while((pkt = ifqueue->filter(ih->saddr())))
            {
                drop(pkt, DROP_RTR_MAC_CALLBACK);
            }

            // if precursor list non-empty add to RERR and delete the
            // precursor list
            if (!rt->pc_empty())
            {
                nre->unreachable_dst[nre->DestCount] = rt->rt_dst;
                nre->unreachable_dst_seqno[nre->DestCount] = rt->rt_seqno;
                nre->DestCount += 1;
                rt->pc_delete();
            }
```

```
        }
    }

    if (nre->DestCount > 0)
    {
        #ifdef DEBUG
        fprintf(stderr, "%s(%f): %d\t sending RERR...\n", __FUNCTION__, NOW, index);
        #endif // DEBUG
        sendError(rerr);
    }
    else
    {
        Packet::free(rerr);
    }

    Packet::free(p);
}
#endif //AOMDV

/*
 Packet Transmission Routines
*/

void AODV::forward(aodv_rt_entry *rt, Packet *p, double delay)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);

    if(ih->ttl_ == 0)
    {
        #ifdef DEBUG
        fprintf(stderr, "%s: calling drop()\n", __PRETTY_FUNCTION__);
        #endif // DEBUG
        drop(p, DROP_RTR_TTL);
        return;
    }

    if (rt)
    {
        assert(rt->rt_flags == RTF_UP);
        ch->addr_type() = NS_AF_INET;

        #ifndef AOMDV
        ch->next_hop() = rt->rt_nexthop;
        rt->rt_expire = NOW + ACTIVE_ROUTE_TIMEOUT;
        // CHANGE
        if ((ih->saddr() != index) && DATA_PACKET(ch->ptype()))
            rt->rt_error = true;
        // CHANGE
        #else //AOMDV

        // [scheduling] ---  changed argument
        #ifdef SA_CHANGE
        AODV_Path *path = rt->path_find(ih->saddr(), index);
        #else
        AODV_Path *path = rt->path_find();
        #endif
        // [scheduling] ---
        #ifdef SA_CHANGE
        if (ih->saddr() == index)
        {
            rt->tx_rx_id = ih->daddr(); // this node is a sender
            // as we assume to have only one outgoing connection this works,
            // otherwise a list of destinations is required
        }
        #endif
        // [scheduling] --- end
```

```
        ch->next_hop() = path->nexthop;
        path->expire = NOW + ACTIVE_ROUTE_TIMEOUT;
        // CHANGE
        if ((ih->saddr() != index) && DATA_PACKET(ch->ptype()))
        {
            rt->rt_error = true;
        }
        // CHANGE
        #endif //AOMDV

        //important: change the packet's direction
        ch->direction() = hdr_cmn::DOWN;

    }

    else  // if it is a broadcast packet
    {
        assert(ih->daddr() == (nsaddr_t) IP_BROADCAST);
        ch->addr_type() = NS_AF_NONE;
        //important: change the packet's direction
        ch->direction() = hdr_cmn::DOWN;
    }

    // If it is a broadcast packet
    if (ih->daddr() == (nsaddr_t) IP_BROADCAST)
    {
        assert(rt == 0);
        /*
        * Jitter the sending of broadcast packets by 10ms
        */
        Scheduler::instance().schedule(target_, p, 0.01 * Random::uniform());
    }

    // Not a broadcast packet
    else
    {
        if(delay > 0.0)
        {
            Scheduler::instance().schedule(target_, p, delay);
        }
        else
        {
            // Not a broadcast packet, no delay, send immediately
            Scheduler::instance().schedule(target_, p, 0.);
        }
    }
}

// CHANGE
void AODV::forwardReply(aodv_rt_entry *rt, Packet *p, double delay)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);

    if(ih->ttl_ == 0)
    {

        #ifdef DEBUG
        fprintf(stderr, "%s: calling drop()\n", __PRETTY_FUNCTION__);
        #endif // DEBUG

        drop(p, DROP_RTR_TTL);
        return;
    }

    if (rt)
    {
```

```
        assert(rt->rt_flags == RTF_UP);
        ch->addr_type() = NS_AF_INET;

        #ifndef AOMDV
        ch->next_hop() = rt->rt_nexthop;
        rt->rt_expire = NOW + ACTIVE_ROUTE_TIMEOUT;
        // CHANGE
        if ((ih->saddr() != index) && DATA_PACKET(ch->ptype()))
            rt->rt_error = true;
        // CHANGE
        #else //AOMDV

        // CHANGE
        if ((ih->saddr() != index) && DATA_PACKET(ch->ptype()))
        {
            rt->rt_error = true;
        }
        // CHANGE
        #endif //AOMDV

        ch->direction() = hdr_cmn::DOWN;
        //important: change the packet's direction
    }
    else  // if it is a broadcast packet
    {
        assert(ih->daddr() == (nsaddr_t) IP_BROADCAST);
        ch->addr_type() = NS_AF_NONE;
        ch->direction() = hdr_cmn::DOWN;
        //important: change the packet's direction
    }

    if (ih->daddr() == (nsaddr_t) IP_BROADCAST)
    {
        // If it is a broadcast packet
        assert(rt == 0);
        /*
        * Jitter the sending of broadcast packets by 10ms
        */
        Scheduler::instance().schedule(target_, p, 0.01 * Random::uniform());
    }
    else  // Not a broadcast packet
    {
        if(delay > 0.0)
        {
            Scheduler::instance().schedule(target_, p, delay);
        }
        else
        {
        // Not a broadcast packet, no delay, send immediately
            Scheduler::instance().schedule(target_, p, 0.);
        }
    }

}
// CHANGE

void AODV::sendRequest(nsaddr_t dst)
{
    // Allocate a RREQ packet
    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
    aodv_rt_entry *rt = rtable.rt_lookup(dst);

    assert(rt);
```

```
/*
* Rate limit sending of Route Requests. We are very conservative
* about sending out route requests.
*/

if (rt->rt_flags == RTF_UP)
{
    assert(rt->rt_hops != INFINITY2);
    Packet::free((Packet *)p);
    return;
}

if (rt->rt_req_timeout > NOW)
{
    Packet::free((Packet *)p);
    return;
}

// rt_req_cnt is the no. of times we did network-wide broadcast
// RREQ_RETRIES is the maximum number we will allow before
// going to a long timeout.

if (rt->rt_req_cnt > RREQ_RETRIES)
{
    rt->rt_req_timeout = NOW + MAX_RREQ_TIMEOUT;
    rt->rt_req_cnt = 0;
    Packet *buf_pkt;
    while ((buf_pkt = rqueue.deque(rt->rt_dst)))
    {
        drop(buf_pkt, DROP_RTR_NO_ROUTE);
    }
    Packet::free((Packet *)p);
    return;
}

#ifdef DEBUG
fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d\n",
        ++route_request, index, rt->rt_dst);
#endif // DEBUG

// Determine the TTL to be used this time.
// Dynamic TTL evaluation - SRD

rt->rt_req_last_ttl = max(rt->rt_req_last_ttl,rt->rt_last_hop_count);

if (0 == rt->rt_req_last_ttl)
{
    // first time query broadcast
    ih->ttl_ = TTL_START;
}
else
{
    // Expanding ring search.
    if (rt->rt_req_last_ttl < TTL_THRESHOLD)
        ih->ttl_ = rt->rt_req_last_ttl + TTL_INCREMENT;
    else
    {
        // network-wide broadcast
        ih->ttl_ = NETWORK_DIAMETER;
        rt->rt_req_cnt += 1;
    }
}

// remember the TTL used for the next time
rt->rt_req_last_ttl = ih->ttl_;

// PerHopTime is the roundtrip time per hop for route requests.
```

```
    // The factor 2.0 is just to be safe .. SRD 5/22/99
    // Also note that we are making timeouts to be larger if we have
    // done network wide broadcast before.

    rt->rt_req_timeout = 2.0 * (double) ih->ttl_ * PerHopTime(rt);
    if (rt->rt_req_cnt > 0)
        rt->rt_req_timeout *= rt->rt_req_cnt;
    rt->rt_req_timeout += NOW;

    // Don't let the timeout to be too large, however .. SRD 6/8/99
    if (rt->rt_req_timeout > NOW + MAX_RREQ_TIMEOUT)
        rt->rt_req_timeout = NOW + MAX_RREQ_TIMEOUT;
    rt->rt_expire = 0;      //init value ? RG

    #ifdef DEBUG
    fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d, tout %f ms\n",
        ++route_request, index, rt->rt_dst, rt->rt_req_timeout - NOW);
    #endif // DEBUG


    // Fill out the RREQ packet
    ch->ptype() = PT_AODV;
    ch->size() = IP_HDR_LEN + rq->size();
    ch->iface() = -2;
    ch->error() = 0;
    ch->addr_type() = NS_AF_NONE;
    ch->prev_hop_ = index; // AODV hack

    ih->saddr() = index;
    ih->daddr() = IP_BROADCAST;
    ih->sport() = RT_PORT;
    ih->dport() = RT_PORT;

    // Fill up some more fields.
    rq->rq_type = AODVTYPE_RREQ;
    rq->rq_hop_count = 0; // CHANGE (previously 1)
    rq->rq_bcast_id = bid++;
    rq->rq_dst = dst;
    rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
    rq->rq_src = index;
    seqno += 2;
    assert ((seqno % 2) == 0);
    rq->rq_src_seqno = seqno;
    rq->rq_timestamp = NOW;

    Scheduler::instance().schedule(target_, p, 0.);

}

void AODV::sendError(Packet *p, bool jitter)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_error *re = HDR_AODV_ERROR(p);

    #ifdef ERROR
    fprintf(stderr, "sending Error from %d at %.2f\n", index, NOW);
    #endif // DEBUG

    re->re_type = AODVTYPE_RERR;
    // DestCount and list of unreachable destinations are already filled

    ch->ptype()     = PT_AODV;
    ch->size()      = IP_HDR_LEN + re->size();
    ch->iface()     = -2;
    ch->error()     = 0;
    ch->addr_type() = NS_AF_NONE;
```

```
    ch->next_hop_    = 0;
    ch->prev_hop_    = index;            // AODV hack
    ch->direction() = hdr_cmn::DOWN;     //important: change the packet's direction

    ih->saddr()      = index;
    ih->daddr()      = IP_BROADCAST;
    ih->sport()      = RT_PORT;
    ih->dport()      = RT_PORT;
    ih->ttl_         = 1;

    // Do we need any jitter? Yes
    if (jitter)
        Scheduler::instance().schedule(target_, p, 0.01*Random::uniform());
    else
        Scheduler::instance().schedule(target_, p, 0.0);

}

// [scheduling] ----
#ifdef SA_CHANGE
#ifdef AODV_PING
void AODV::sendPing()
{
//ping all the destinations via all paths
    aodv_rt_entry *rt, *rtn;
    double delay;
    // run through the routing table and create a ping for all paths
    // for each rt entry
    for(rt = rtable.head(); rt; rt = rtn)
    {
        // check if route is up and check the dest
        if (rt->rt_flags == RTF_UP && rt->rt_dst ==rt->tx_rx_id )
        {
            AODV_Path *path = rt->rt_path_list.lh_first;
            for(; path; path = path->path_link.le_next)
            // all paths
            {
                delay = 0.01 * Random::uniform();
                if (path->expire > (NOW + delay))
                {
                    Packet *p = Packet::alloc();
                    struct hdr_cmn *ch = HDR_CMN(p);
                    struct hdr_ip *ih = HDR_IP(p);
                    struct hdr_aodv_ping *ping = HDR_AODV_PING(p);

                    ping->type      = AODVTYPE_PING;
                    ping->src       = index;            // src of ping
                    ping->dst       = rt->rt_dst;       // dst of ping
                    ping->last_hop  = path->lasthop;
                    ping->first_hop = path->nexthop;
                    ping->hop_count = 0;
                    ping->ping_reply_flag = 0;
                    ping->nr_neigh  = 0;
                    ping->next_hop  = path->nexthop;    // jump by jump

                    ch->ptype()     = PT_AODV;
                    ch->size()      = IP_HDR_LEN + ping->size();
                    ch->iface()     = -2;
                    ch->error()     = 0;
                    ch->addr_type() = NS_AF_INET;
                    ch->prev_hop_   = index;
                    ch->timestamp() = NOW + delay;      // set the time
                    ch->next_hop()  = path->nexthop;
                    ch->direction() = hdr_cmn::DOWN;    // down to phy

//                  ih->daddr()     = path->nexthop;    // jump by jump
                    ih->daddr()     = rt->rt_dst;
```

```
                    ih->saddr()      = index;
                    ih->sport()      = RT_PORT;
                    ih->dport()      = RT_PORT;
                    ih->ttl_         = NETWORK_DIAMETER;

                    path->expire     = NOW + ACTIVE_PATH_TIMEOUT;     // set new timeout
                    rt->rt_expire    = NOW + ACTIVE_ROUTE_TIMEOUT;    // set new timeout

                    #ifdef SA_DEBUG
                    fprintf(stdout, "Ping from [%d] at t = %.7f via \%d to [%d] \n",
                            index, ch->timestamp(), path->nexthop, rt->rt_dst);
                    #endif

                    // dont send all the packets at the same time, but
                    // this has to be taken
                    // into account for the timestamp !!
                    Scheduler::instance().schedule(target_, p, delay);
                }
            }
        }
        rtn = rt->rt_link.le_next;
    }
}

void AODV::sendPong(nsaddr_t dest, Packet *p)
{
    // create a ping response
    Packet *pong = Packet::alloc();              // new packet
    struct hdr_cmn  *ch = HDR_CMN(pong);
    struct hdr_ip   *ih = HDR_IP(pong);
    struct hdr_aodv_ping *ping = HDR_AODV_PING(pong);

    struct hdr_cmn  *ch_s = HDR_CMN(p);          // use the old values to initialise
    struct hdr_ip   *ih_s = HDR_IP(p);
    struct hdr_aodv_ping *ping_s = HDR_AODV_PING(p);


    // find the route entry
    aodv_rt_entry *rt;
    rt = rtable.rt_lookup(ping_s->src);
    if (rt != NULL)
    {
        AODV_Path *path ;
        path = rt->path_lookup(ping_s->last_hop);

        if (path != NULL)
        {
            path->expire = NOW + ACTIVE_PATH_TIMEOUT;   // set new timeout
            rt->rt_expire = NOW + ACTIVE_ROUTE_TIMEOUT; // set new timeout
        }
        else
        {
            #ifdef SA_DEBUG
            fprintf(stdout, "no path to return ping");
            #endif
        }
    }
    else
    {
            #ifdef SA_DEBUG
            fprintf(stdout, "no route to return ping");
            #endif
    }

    ping->type      = AODVTYPE_PING;
    ping->src       = ping_s->src;
    ping->dst       = ping_s->dst;               // do not overwrite the source and dest
```

```
    ping->first_hop = ping_s->last_hop;
    ping->last_hop  = ping_s->first_hop;
    ping->hop_count = ping_s->hop_count;
    ping->ping_reply_flag = 1;
    ping->nr_neigh  = ping_s->nr_neigh;
    ping->next_hop  = dest;

    ch->ptype()     = PT_AODV;
    ch->size()      = IP_HDR_LEN + ping->size();

    ch->iface()     = -2;
    ch->error()     = 0;
    ch->addr_type() = NS_AF_INET;
    ch->prev_hop_   = index;
    ch->timestamp() = ch_s->timestamp();
    ch->next_hop()  = ch_s->prev_hop_;
    ch->direction() = hdr_cmn::DOWN;

    ih->saddr()     = index;
    ih->daddr()     = ping_s->src;
    ih->sport()     = RT_PORT;
    ih->dport()     = RT_PORT;
    ih->ttl_        = ih_s->ttl_;

    Packet::free(p);                // delete the old packet
    #ifdef SA_DEBUG
    fprintf(stdout, "Pong from [%d] at t = %.7f via \%d to [%d] \n",
        index,ch->timestamp(), ch->next_hop(), ping->src);
    #endif

    Scheduler::instance().schedule(target_, pong, 0);
}

void AODV::recvPing(Packet *p)
{// receive the ping packet
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_aodv_ping *ping = HDR_AODV_PING(p);

    AODV_Neighbor *nb;
    // check ttl
    if(ih->ttl_ == 0)
    {
        #ifdef SA_DEBUG
        fprintf(stdout, "drop %d: at node %d from %d ttl %d\n", ih->flowid(),
            index, ping->dst, ih->ttl());
        #endif
        drop(p, DROP_RTR_TTL);
        return;
    }

    /*************************************/
    // AT SOURCE
    /*************************************/
    if (ping->src == index)
    {

        // should not occur, as we do not want to receive our own packets
        if (ping->ping_reply_flag == 0)
        {
            #ifdef SA_DEBUG
            fprintf(stdout, "received my own ping!\n");
            #endif
            drop(p, DROP_RTR_NO_ROUTE);
            return;
        }
```

```
    // ping came back to origin
    else
    {
        double rtt;
        rtt = NOW - ch->timestamp() ;
        #ifdef SA_DEBUG
        fprintf(stdout, "ping had rtt %f from [%d] to node [%d]\n",
            rtt, index, ping->dst);
        #endif
        aodv_rt_entry* entry;
        entry = rtable.rt_lookup(ping->dst);
        if (entry == NULL)
        // no rt entry
        {
            #ifdef SA_DEBUG
            fprintf(stdout, "ping to a dest. that does not exist\n");
            #endif
            drop(p, DROP_RTR_NO_ROUTE);
            return;
        }
        else
        {
            AODV_Path* path;
            path = entry->path_lookup(ping->last_hop);
            // returns the path with same first hop
            if (path == NULL)
            {
                #ifdef SA_DEBUG
                fprintf(stdout, "pinged path already deleted\n");
                rtable.rt_dumptable();
                #endif
                drop(p, DROP_RTR_NO_ROUTE);
                return;
            }
            else
            {
                path->rtt = rtt;
                path->expire = NOW + ACTIVE_PATH_TIMEOUT;
                // update lifetime
                if (ping->nr_neigh != 0)
                    path->nr_neigh = ping->nr_neigh;
                entry->sched_calc_weights();
                Packet::free(p);
                return;
            }
        }
    }
}

/*************************************/
// AT DESTINATION
/*************************************/
else if (ping->dst == index)
{
    #ifdef SA_DEBUG
    cout << "received Ping at node " << index << " from " << ih->saddr()
        << " ttl "<< ih->ttl() <<"\n";
    # endif
    sendPong(ch->prev_hop_, p); // via the hop it came in
    return;
}

/*************************************/
// FORWARD
/*************************************/
else
{
```

```
// find the route entry
aodv_rt_entry* entry;
if ( ping->ping_reply_flag == 0 )   // this is a ping message
{
    entry = rtable.rt_lookup(ping->dst);
}
else                                // if it is a pong
{
    entry = rtable.rt_lookup(ping->src);
}

if (entry == NULL)
{
    drop(p, DROP_RTR_NO_ROUTE);
    return;
}
else
{
    entry->rt_expire = NOW + ACTIVE_ROUTE_TIMEOUT;
    // update the lifetime
    // find the path entry
    AODV_Path* path = entry->rt_path_list.lh_first;
    for(; path; path = path->path_link.le_next)
    {
        if (path->lasthop == ping->last_hop)
        // search path with same last hop
            break;
    }
    if (path == NULL)
    {
        drop(p, DROP_RTR_NO_ROUTE);
        return;
    }
    else
    {
        path->expire    = NOW + ACTIVE_PATH_TIMEOUT;    // update lifetime
        // set the new dest
        ping->next_hop  = path->nexthop;
        ch->next_hop()  = path->nexthop;
        ch->prev_hop_   = index;
        ch->direction() = hdr_cmn::DOWN;

        if (ping->ping_reply_flag == 0)     // this is a ping message
        {
            path->ping_sent = NOW;
            // insert the neighbours of this node
            AODV_Neighbor *neigh;
            neigh = nbhead.lh_first;
            // count the neighbours
            int count = 0;
            for (; neigh; neigh = (neigh->nb_link.le_next ))
            {
                count ++;
            }
            ping->nr_neigh += count;

        }
        else
        {
            // store the rtt from here to the destination
            entry = rtable.rt_lookup(ping->dst);
            if (entry != NULL)
            {
                path = entry->path_lookup(ping->dst);
                if (path != NULL)
                {
                    path->rtt = NOW - path->ping_sent;
```

```
                        }
                    }
                }
                #ifdef SA_DEBUG
                cout << "at node "<< index << " forward Ping to "  << ping->dst <<"\n";
                #endif
                Scheduler::instance().schedule(target_, p, 0);
            }
        }
    }
}
#endif //PING

#ifdef AODV_INFO
void AODV::sendInfo()
{ // print informations to the trace file using the info packet
    Packet *p = Packet::alloc();

    struct hdr_cmn *ch = HDR_CMN(p);         // the common header
    struct hdr_ip *ih   = HDR_IP(p);         // the IP header
    struct hdr_aodv_info *info = HDR_AODV_INFO(p);

    info->type      = AODVTYPE_INFO;
    double delay;
    info->src       = node_info.src;
    info->dst       = node_info.dst;
    info->action    = node_info.action;
    info->path_count= node_info.path_count;
    info->next_hop  = node_info.next_hop;
    info->last_hop  = node_info.last_hop;
    info->hop_count = node_info.hop_count;
    info->rtt       = node_info.rtt;

    ch->ptype()     = PT_AODV;
    ch->iface()     = -2;
    ch->error()     = 0;
    ch->addr_type() = NS_AF_INET;
    ch->prev_hop_   = index;
    ch->direction() = hdr_cmn::DOWN;         // down to phy

    ih->saddr()     = index;
    ih->daddr()     = -1;
    ih->sport()     = RT_PORT;
    ih->dport()     = RT_PORT;
    ih->ttl_        = 0;                     // dont send it

    drop(p);
}

void AODV::dumpPath( aodv_rt_entry *rt, AODV_Path *path, int action, int num_path)
{ // collect all the informations we want to print out

    node_info.src       = index;
    node_info.dst       = rt->rt_dst;
    node_info.path_count= num_path;
    node_info.action    = action;
    node_info.next_hop  = path->nexthop;
    node_info.last_hop  = path->lasthop;
    node_info.hop_count = path->hopcount;
//  node_info.rtt       = path->rtt;
    node_info.rtt       = path->expire;

    sendInfo();

}
#endif //INFO
#endif
```

```
// [scheduling] ---- end

/*
 Neighbor Management Functions
*/

void AODV::sendHello()
{
    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_reply *rh = HDR_AODV_REPLY(p);

    #ifdef DEBUG
    fprintf(stderr, "sending Hello from %d at %.2f\n", index, NOW);
    #endif // DEBUG

    rh->rp_type      = AODVTYPE_HELLO;
    rh->rp_hop_count= 0;
    rh->rp_dst       = index;
    rh->rp_dst_seqno= seqno;
    rh->rp_lifetime = (1 + ALLOWED_HELLO_LOSS) * HELLO_INTERVAL;

    ch->ptype()      = PT_AODV;
    ch->size()       = IP_HDR_LEN + rh->size();
    ch->iface()      = -2;
    ch->error()      = 0;
    ch->addr_type() = NS_AF_NONE;
    ch->prev_hop_    = index; // AODV hack

    ih->saddr()      = index;
    ih->daddr()      = IP_BROADCAST;
    ih->sport()      = RT_PORT;
    ih->dport()      = RT_PORT;
    ih->ttl_     = 1;

    Scheduler::instance().schedule(target_, p, 0.0);
}

void AODV::recvHello(Packet *p)
{
    struct hdr_ip *ih = HDR_IP(p); // CHANGED BY ME (uncommented this line)
    struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
    AODV_Neighbor *nb;

    nb = nb_lookup(rp->rp_dst);
    if(nb == 0)
    {
        nb_insert(rp->rp_dst);
    }
    else
    {
        nb->nb_expire = NOW + (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
    }

    // CHANGE
    // Add a route to this neighbor
    ih->daddr() = index;
    rp->rp_src = ih->saddr();
    #ifdef AOMDV
    rp->rp_first_hop = index;
    #endif // AOMDV
    recvReply(p);
    // CHANGE

}
```

```
void AODV::nb_insert(nsaddr_t id)
{
    // CHANGE
    AODV_Neighbor *nb;
    if ( ( nb=nb_lookup(id) ) == NULL)
    {
        nb = new AODV_Neighbor(id);
        assert(nb);
        // CHANGE
        nb->nb_expire = NOW + (HELLO_INTERVAL * ALLOWED_HELLO_LOSS);

        LIST_INSERT_HEAD(&nbhead, nb, nb_link);
    }
    else
    {
        // CHANGE
        nb->nb_expire = NOW + (HELLO_INTERVAL * ALLOWED_HELLO_LOSS);
    }

}

AODV_Neighbor* AODV::nb_lookup(nsaddr_t id)
{
    AODV_Neighbor *nb = nbhead.lh_first;

    for(; nb; nb = nb->nb_link.le_next)
    {
        if(nb->nb_addr == id) break;
    }
    return nb;
}

/*
 * Called when we receive *explicit* notification that a Neighbor
 * is no longer reachable.
 */
void AODV::nb_delete(nsaddr_t id)
{
    AODV_Neighbor *nb = nbhead.lh_first;

    log_link_del(id);
    seqno += 2; // Set of neighbors changed
    assert ((seqno % 2) == 0);

    for(; nb; nb = nb->nb_link.le_next)
    {
        if(nb->nb_addr == id)
        {
            LIST_REMOVE(nb,nb_link);
            delete nb;
            break;
        }
    }

    Packet *p;

    handle_link_failure(id);

    #ifdef AOMDV_PACKET_SALVAGING
    while((p = ifqueue->filter(id)))
    {
        struct hdr_cmn *ch = HDR_CMN(p);
        struct hdr_ip *ih = HDR_IP(p);

        if ( !DATA_PACKET(ch->ptype()) ) drop(p, DROP_RTR_HELLO);
        else
        {
```

```
            // salvage the packet using an alternate path if available.
            aodv_rt_entry *rt = rtable.rt_lookup(ih->daddr());
            if ( rt && (rt->rt_flags == RTF_UP)  &&
                (ch->aomdv_salvage_count_ < AOMDV_MAX_SALVAGE_COUNT) )
            {
                ch->aomdv_salvage_count_ += 1;
                forward(rt, p, NO_DELAY);
            }
            else drop(p, DROP_RTR_HELLO);
        }
    }

    #else // NO PACKET SALVAGING

    while((p = ifqueue->filter(id)))
    {
        drop(p, DROP_RTR_HELLO);
    }

    #endif // NO PACKET SALVAGING
}

/*
 * Purges all timed-out Neighbor Entries - runs every
 * HELLO_INTERVAL * 1.5 seconds.
 */
void AODV::nb_purge()
{
    AODV_Neighbor *nb = nbhead.lh_first;
    AODV_Neighbor *nbn;

    for(; nb; nb = nbn)
    {
        nbn = nb->nb_link.le_next;
        if(nb->nb_expire <= NOW)
        {
            nb_delete(nb->nb_addr);
        }
    }

}

#ifdef AOMDV
void AODV::recvRequest(Packet *p)
{
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
    aodv_rt_entry *rto, *rt;
    BroadcastID* b = NULL;
    bool kill_request_propagation = false;
    AODV_Path* reverse_path = NULL;

    /* If I have received my own RREQ - just drop it. */
    if(rq->rq_src == index)
    {
        Packet::free(p);
        return;
    }
    /* If RREQ has already been received - drop it, else remember
    "RREQ id" <src IP, bcast ID>. */
    if ( (b = id_lookup(rq->rq_src, rq->rq_bcast_id)) == NULL)
    {
        // Cache the broadcast ID
        b = id_insert(rq->rq_src, rq->rq_bcast_id);
    }
    else
        kill_request_propagation = true;
```

```
/* If I am a neighbor to the RREQ source, make myself first hop on path from
source to dest. */
if (rq->rq_hop_count == 0)
    rq->rq_first_hop = index;

/* Check if an entry for RREQ source exists already in route table */
rto = rtable.rt_lookup(rq->rq_src);
if(rto == 0)
{
/* if entry not in the route table create an entry for the reverse route. */
    rto = rtable.rt_add(rq->rq_src);
}

/*
* Create/update reverse path (i.e. path back to RREQ source)
* If RREQ contains more recent seq number than route table entry - update route
* entry to source.
*/
if (rto->rt_seqno < rq->rq_src_seqno)
{
    rto->rt_seqno = rq->rq_src_seqno;
    rto->rt_advertised_hops = INFINITY;
    // [scheduling] ---
    #ifdef SA_CHANGE
    // print the paths before they are removed
    #ifdef AODV_INFO
    AODV_Path *path = rto->rt_path_list.lh_first;
    int num_path = rto->rt_num_paths_;
    if (rto->path_empty() == false)
    {
        for(; path; path = path->path_link.le_next)
        {
            num_path --;
            dumpPath(rto, path, -1, num_path);
        }
    }
    #endif
    #endif
    // [scheduling] --- end
    rto->path_delete(); // Delete all previous paths to RREQ source

    rto->rt_flags = RTF_UP;
    /* Insert new path for route entry to source of RREQ.
    (src addr, hop count + 1, lifetime, last hop (first hop for RREQ)) */
    reverse_path = rto->path_insert(ih->saddr(), rq->rq_hop_count+1,
            NOW + REV_ROUTE_LIFE, rq->rq_first_hop);

    // [scheduling] ---
    #ifdef SA_CHANGE
    #ifdef AODV_INFO
    dumpPath(rto, reverse_path, 1, rto->rt_num_paths_);
    #endif
    #endif
    // [scheduling] --- end

    // CHANGE
    rto->rt_last_hop_count = rto->path_get_max_hopcount();
    // CHANGE
}
/* If a new path with smaller hop count is received
(same seqno, better hop count) - try to insert new path in route table. */
else if ( (rto->rt_seqno == rq->rq_src_seqno) &&
      (rto->rt_advertised_hops > rq->rq_hop_count) )
{
    AODV_Path* erp=NULL;
```

```
        assert(rto->rt_flags == RTF_UP); // Make sure path is up

        /*
        * If path already exists - adjust the lifetime of the path.
        */
        if ((reverse_path = rto->disjoint_path_lookup(ih->saddr(),
            rq->rq_first_hop)))
        {
            assert(reverse_path->hopcount == (rq->rq_hop_count+1));
            reverse_path->expire = max(reverse_path->expire, (NOW + REV_ROUTE_LIFE));
        }
        /*
        * Got a new alternate disjoint reverse path - so insert it.
        * I.e. no path exists which has RREQ source as next hop and no
        * path with RREQ first hop as last hop exists for this route entry.
        * Simply stated: no path with the same last hop exists already.
        */
        else if (rto->new_disjoint_path(ih->saddr(), rq->rq_first_hop))
        {
            /* Only insert new path if not too many paths exists for this
             * destination  and new path does not differ too much in length
             * compared to previous paths */
            if ( (rto->rt_num_paths_ < aomdv_max_paths_) &&
               (((rq->rq_hop_count + 1) - rto->path_get_min_hopcount()) <=
                aomdv_prim_alt_path_len_diff_) )
            {
                /* Insert new (disjoint) reverse path */
                reverse_path = rto->path_insert(ih->saddr(),
                        rq->rq_hop_count+1, NOW + REV_ROUTE_LIFE,
                        rq->rq_first_hop);

                // [scheduling] ---
                #ifdef SA_CHANGE
                #ifdef AODV_INFO
                dumpPath(rto, reverse_path, 1, rto->rt_num_paths_);
                #endif
                #endif
                // [scheduling] --- end

                // CHANGE
                rto->rt_last_hop_count = rto->path_get_max_hopcount();
                // CHANGE
            }
            /* If new path differs too much in length compared to previous
             * paths - drop packet. */
            if (((rq->rq_hop_count + 1) - rto->path_get_min_hopcount()) >
                aomdv_prim_alt_path_len_diff_)
            {
                Packet::free(p);
                return;
            }
        }
        /* (RREQ was intended for me) AND
        ((Path with RREQ first hop as last hop does not exist) OR
        (The path exists and has less hop count than RREQ)) - drop packet.
        Don't know what this case is for... */
        else if ( (rq->rq_dst == index) &&
            ( ((erp = rto->path_lookup_lasthop(rq->rq_first_hop)) == NULL) ||
            ((rq->rq_hop_count+1) > erp->hopcount) ) )
        {
            Packet::free(p);
            return;
        }
    }
    /* Older seqno (or same seqno with higher hopcount), i.e. I have a
    more recent route entry - so drop packet.*/
    else
```

```
{
    Packet::free(p);
    return;
}

/* If route is up */
if (rto->rt_flags == RTF_UP)
{
    // Reset the soft state
    rto->rt_req_timeout = 0.0;
    rto->rt_req_last_ttl = 0;
    rto->rt_req_cnt = 0;

    /*
    * Find out whether any buffered packet can benefit from the
    * reverse route.
    */
    Packet *buffered_pkt;
    while ((buffered_pkt = rqueue.deque(rto->rt_dst)))
    {
        if (rto && (rto->rt_flags == RTF_UP))
        {
            forward(rto, buffered_pkt, NO_DELAY);
        }
    }
}
/* Check route entry for RREQ destination */
rt = rtable.rt_lookup(rq->rq_dst);

/* I am the intended receiver of the RREQ - so send a RREP */
if (rq->rq_dst == index)
{
    if (seqno < rq->rq_dst_seqno)
    {
        seqno = rq->rq_dst_seqno + 1;
    }
    /* Make sure seq number is even (why?) */
    if (seqno % 2)
        seqno++;


    sendReply(  rq->rq_src,      // IP Destination
            0,                   // Hop Count
            index,               // (RREQ) Dest IP Address
            seqno,               // Dest Sequence Num
            MY_ROUTE_TIMEOUT,    // Lifetime
            rq->rq_timestamp,    // timestamp
            ih->saddr(),         // nexthop
            rq->rq_bcast_id,     // broadcast id to identify this route discovery
            ih->saddr());

    Packet::free(p);
}
/* I have a fresh route entry for RREQ destination - so send RREP */
else if ( rt && (rt->rt_flags == RTF_UP) && (rt->rt_seqno >= rq->rq_dst_seqno) )
{
    assert ((rt->rt_seqno % 2) == 0); // is the seqno even?
    /* Reverse path exists */
    if (reverse_path)
    {
        #ifdef AOMDV_NODE_DISJOINT_PATHS
        if (b->count == 0)
        {
            b->count = 1;

            // route advertisement
            if (rt->rt_advertised_hops == INFINITY)
```

```
            rt->rt_advertised_hops = rt->path_get_max_hopcount();
            #ifdef SA_CHANGE
            AODV_Path *forward_path = rt->path_find(ih->saddr(), index);
            #else
            AODV_Path *forward_path = rt->path_find();
            #endif
            // [scheduling] argument
            // CHANGE
            rt->rt_error = true;
            // CHANGE
            sendReply(rq->rq_src,
                rt->rt_advertised_hops,
                rq->rq_dst,
                rt->rt_seqno,
                forward_path->expire - NOW,
                rq->rq_timestamp,
                ih->saddr(),
                rq->rq_bcast_id,
                forward_path->lasthop);
        }
        #endif //AOMDV_NODE_DISJOINT_PATHS
        #ifdef AOMDV_LINK_DISJOINT_PATHS

        AODV_Path* forward_path = NULL;
        AODV_Path *r = rt->rt_path_list.lh_first;
        // Get first path for RREQ destination
        /* Make sure we don't answer with the same forward path twice in
        response  to a certain RREQ (received more than once). E.g.
        "middle node"   in "double diamond". */
        for(; r; r = r->path_link.le_next)
        {
            if (b->forward_path_lookup(r->nexthop, r->lasthop)== NULL)
            {
                forward_path = r;
                break;
            }
        }

        /* If an unused forward path is found and we have not answered
        along this reverse path (for this RREQ) - send a RREP back. */
        if ( forward_path &&(b->reverse_path_lookup(reverse_path->nexthop,
            reverse_path->lasthop) == NULL) )
        {
            /* Mark the reverse and forward path as used (for this
            RREQ). */
            b->reverse_path_insert(reverse_path->nexthop, reverse_path->lasthop);
            b->forward_path_insert(forward_path->nexthop, forward_path->lasthop);

            // route advertisement
            if (rt->rt_advertised_hops == INFINITY)
                rt->rt_advertised_hops = rt->path_get_max_hopcount();

            // CHANGE
            rt->rt_error = true;
            // CHANGE
            sendReply(rq->rq_src,
                 rt->rt_advertised_hops,
                 rq->rq_dst,
                 rt->rt_seqno,
                 forward_path->expire - NOW,
                 rq->rq_timestamp,
                 ih->saddr(),
                 rq->rq_bcast_id,
                 forward_path->lasthop);
        }
        #endif //AOMDV_LINK_DISJOINT_PATHS
    }
```

```
        Packet::free(p);
    }
    /* RREQ not intended for me and I don't have a fresh
    enough entry for RREQ dest - so forward the RREQ */
    else
    {
        if (kill_request_propagation)
        {
            // do not propagate a duplicate RREQ
            Packet::free(p);
            return;
        }
        else
        {
            ih->saddr() = index;

            // Maximum sequence number seen en route
            if (rt)
                rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);

            // route advertisement
            if (rto->rt_advertised_hops == INFINITY)
                rto->rt_advertised_hops = rto->path_get_max_hopcount();
            rq->rq_hop_count = rto->rt_advertised_hops;
            #ifdef AOMDV_NODE_DISJOINT_PATHS
            #ifdef SA_CHANGE
            rq->rq_first_hop = (rto->path_find(ih->saddr(), index))->lasthop;
            #else
            rq->rq_first_hop = (rto->path_find())->lasthop;
            #endif
            // [scheduling] argument
            #endif //AOMDV_NODE_DISJOINT_PATHS

            forward((aodv_rt_entry*) o, p, DELAY);
        }
    }
}

void AODV::sendReply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst,
 u_int32_t rpseq, u_int32_t lifetime, double timestamp,
 nsaddr_t nexthop, u_int32_t bcast_id, nsaddr_t rp_first_hop)
{
    Packet *p = Packet::alloc();
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);

    #ifdef DEBUG
    fprintf(stderr, "sending Reply from %d at %.2f\n", index, NOW);
    #endif // DEBUG

    rp->rp_type      = AODVTYPE_RREP;
    rp->rp_hop_count= hop_count;
    rp->rp_dst       = rpdst;
    rp->rp_dst_seqno= rpseq;
    rp->rp_src       = index;
    rp->rp_lifetime = lifetime;
    rp->rp_timestamp= timestamp;
    rp->rp_bcast_id = bcast_id;
    rp->rp_first_hop= rp_first_hop;

    ch->ptype()      = PT_AODV;
    ch->size()       = IP_HDR_LEN + rp->size();
    ch->iface()      = -2;
    ch->error()      = 0;
    ch->addr_type() = AF_INET;
```

```
    ch->next_hop_    = nexthop;

    ch->xmit_failure_ = aodv_rt_failed_callback;
    ch->xmit_failure_data_ = (void*) this;

    ih->saddr()      = index;
    ih->daddr()      = ipdst;
    ih->sport()      = RT_PORT;
    ih->dport()      = RT_PORT;
    ih->ttl_         = NETWORK_DIAMETER;

    Scheduler::instance().schedule(target_, p, 0.);
}

void AODV::recvReply(Packet *p)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
    aodv_rt_entry *rto, *rt;
    BroadcastID* b = NULL;
    AODV_Path* forward_path = NULL;

    /* If I receive a RREP with myself as source - drop packet (should not occur).
    Comment: rp_dst is the source of the RREP, or rather the destination of the RREQ.
    */
    if (rp->rp_dst == index)
    {
        Packet::free(p);
        return;
    }
    /* Check routing table for a path to (RREQ) destination */
    rt = rtable.rt_lookup(rp->rp_dst);
    /* If a path to (RREQ) destination does not exist already we add a
    forward path entry to the (RREQ) destination */
    if(rt == 0)
    {
        rt = rtable.rt_add(rp->rp_dst);
    }
    /* If RREP contains more recent seqno for (RREQ) destination -
    delete all old paths and add the new forward path to (RREQ) destination */
    if (rt->rt_seqno < rp->rp_dst_seqno)
    {
        rt->rt_seqno = rp->rp_dst_seqno;
        rt->rt_advertised_hops = INFINITY;
        // [scheduling] ---
        #ifdef SA_CHANGE
        // remove all paths
        #ifdef AODV_INFO
        AODV_Path *path = rt->rt_path_list.lh_first;
        int num_path = rt->rt_num_paths_;
        for(; path; path = path->path_link.le_next)
        {
            num_path--;
            dumpPath(rt, path, -1, num_path);
        }
        #endif
        #endif
        // [scheduling] --- end
        rt->path_delete();

        rt->rt_flags = RTF_UP;
        /* Insert forward path to RREQ destination. */
        forward_path = rt->path_insert(rp->rp_src, rp->rp_hop_count+1,
                NOW + rp->rp_lifetime, rp->rp_first_hop);

        // [scheduling] ---
```

```
    #ifdef SA_CHANGE
    #ifdef AODV_INFO
    dumpPath(rt, forward_path, 1, rt->rt_num_paths_);
    #endif
    #endif
    // [scheduling] --- end

    // CHANGE
    rt->rt_last_hop_count = rt->path_get_max_hopcount();
    // CHANGE

}
/* If the sequence number in the RREP is the same as for route entry but
with a smaller hop count - try to insert new forward path to (RREQ) dest. */
else if ( (rt->rt_seqno == rp->rp_dst_seqno) &&
    (rt->rt_advertised_hops > rp->rp_hop_count) )
{

    assert (rt->rt_flags == RTF_UP);
    /* If the path already exists - increase path lifetime */
    if ((forward_path = rt->disjoint_path_lookup(rp->rp_src, rp->rp_first_hop)))
    {
        assert (forward_path->hopcount == (rp->rp_hop_count+1));
        forward_path->expire = max(forward_path->expire, NOW + rp->rp_lifetime);
    }
    /* If the path does not already exist, there is room for it and it
    does not differ too much in length - we add the path */
    else if ( rt->new_disjoint_path(rp->rp_src, rp->rp_first_hop) &&
        (rt->rt_num_paths_ < aomdv_max_paths_) &&
        ((rp->rp_hop_count+1) - rt->path_get_min_hopcount() <=
            aomdv_prim_alt_path_len_diff_) )
    {
        /* Insert forward path to RREQ destination. */
        forward_path = rt->path_insert(rp->rp_src, rp->rp_hop_count+1,
                NOW + rp->rp_lifetime, rp->rp_first_hop);
        // CHANGE
        // [scheduling] ---
        #ifdef SA_CHANGE
        #ifdef AODV_INFO
        dumpPath(rt, forward_path, 1, rt->rt_num_paths_);
        #endif
        #endif
        // [scheduling] --- end

        rt->rt_last_hop_count = rt->path_get_max_hopcount();
        // CHANGE

    }
    /* Path did not exist nor could it be added - just drop packet. */
    else
    {
        Packet::free(p);
        return;
    }
}
/* The received RREP did not contain more recent information
than route table - so drop packet */
else
{
    Packet::free(p);
    return;
}
/* If route is up */
if (rt->rt_flags == RTF_UP)
{
    // Reset the soft state
    rt->rt_req_timeout = 0.0;
```

```
        rt->rt_req_last_ttl = 0;
        rt->rt_req_cnt = 0;

        if (ih->daddr() == index)
        {
            // I am the RREP destination

            #ifdef DYNAMIC_RREQ_RETRY_TIMEOUT
            // This macro does not seem to be set.
            if (rp->rp_type == AODVTYPE_RREP)
            {
                rt->rt_disc_latency[rt->hist_indx] = (NOW - rp->rp_timestamp)
                / (double) (rp->rp_hop_count+1);
                // increment indx for next time
                rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
            }
            #endif //DYNAMIC_RREQ_RETRY_TIMEOUT
        }

        /*
        * Find out whether any buffered packet can benefit from the
        * forward route.
        */
        Packet *buffered_pkt;
        while ((buffered_pkt = rqueue.deque(rt->rt_dst)))
        {
            if (rt && (rt->rt_flags == RTF_UP))
            {
                forward(rt, buffered_pkt, NO_DELAY);
            }
        }
    }
    /* If I am the intended receipient of the RREP nothing more needs
    to be done - so drop packet. */
    if (ih->daddr() == index)
    {
        Packet::free(p);
        return;
    }
    /* If I am not the intended receipient of the RREP - check route
    table for a path to the RREP dest (i.e. the RREQ source). */
    rto = rtable.rt_lookup(ih->daddr());
    b = id_lookup(ih->daddr(), rp->rp_bcast_id);
    // Check for <RREQ src IP, bcast ID> tuple

    #ifdef AOMDV_NODE_DISJOINT_PATHS
    if ( (rto == NULL) || (rto->rt_flags != RTF_UP) || (b == NULL) || (b->count) )
    {
        Packet::free(p);
        return;
    }

    b->count = 1;
    #ifdef SA_CHANGE
    AODV_Path *reverse_path = rto->path_find(ih->saddr(), index);
    // [scheduling] argument
    #else
    AODV_Path *reverse_path = rto->path_find();
    #endif
    ch->addr_type() = AF_INET;
    ch->next_hop_ = reverse_path->nexthop;
    ch->xmit_failure_ = aodv_rt_failed_callback;
    ch->xmit_failure_data_ = (void*) this;

    // route advertisement
    rp->rp_src = index;
    if (rt->rt_advertised_hops == INFINITY)
```

```
    rt->rt_advertised_hops = rt->path_get_max_hopcount();
rp->rp_hop_count = rt->rt_advertised_hops;
#ifdef SA_CHANGE
rp->rp_first_hop = (rt->path_find(ih->saddr(), index))->lasthop;
// [scheduling] argument
#else
rp->rp_first_hop = (rt->path_find())->lasthop;
#endif
reverse_path->expire = NOW + ACTIVE_ROUTE_TIMEOUT;

// CHANGE
rt->rt_error = true;
// CHANGE
forward(rto, p, NO_DELAY);

#endif //AOMDV_NODE_DISJOINT_PATHS
#ifdef AOMDV_LINK_DISJOINT_PATHS
/* Drop the RREP packet if we do not have a path back to the source,
or the route is marked as down, or if we never received the original RREQ. */
if ( (rto == NULL) || (rto->rt_flags != RTF_UP) || (b == NULL) )
{
    Packet::free(p);
    return;
}
/* Make sure we don't answer along the same path twice in response
to a certain RREQ. Try to find an unused (reverse) path to forward the RREP. */
AODV_Path* reverse_path = NULL;
AODV_Path *r = rto->rt_path_list.lh_first;
for(; r; r = r->path_link.le_next)
{
    if (b->reverse_path_lookup(r->nexthop, r->lasthop) == NULL)
    {
        reverse_path = r;
        break;
    }
}
/* If an unused reverse path is found and the forward path (for
this RREP) has not already been replied - forward the RREP. */
if ( reverse_path &&
    (b->forward_path_lookup(forward_path->nexthop, forward_path->lasthop) == NULL) )
{
    assert (forward_path->nexthop == rp->rp_src);
    assert (forward_path->lasthop == rp->rp_first_hop);
    /* Mark the forward and reverse path used to answer this RREQ as used. */
    b->reverse_path_insert(reverse_path->nexthop, reverse_path->lasthop);
    b->forward_path_insert(forward_path->nexthop, forward_path->lasthop);

    ch->addr_type() = AF_INET;
    ch->next_hop_ = reverse_path->nexthop;
    ch->xmit_failure_ = aodv_rt_failed_callback;
    ch->xmit_failure_data_ = (void*) this;

    // route advertisement
    if (rt->rt_advertised_hops == INFINITY)
        rt->rt_advertised_hops = rt->path_get_max_hopcount();
    rp->rp_hop_count = rt->rt_advertised_hops;
    rp->rp_src = index;

    reverse_path->expire = NOW + ACTIVE_ROUTE_TIMEOUT;

    // CHANGE
    rt->rt_error = true;
    // CHANGE
    forwardReply(rto, p, NO_DELAY); // CHANGE (previously used forward())

}
else
```

```
    {
        Packet::free(p);
        return;
    }
    #endif //AOMDV_LINK_DISJOINT_PATH

    // [scheduling] ——
    #ifdef SA_CHANGE
    #ifdef AODV_PING
    sendPing();
    #endif
    #endif
    // [scheduling] —— end

}

void AODV::handle_link_failure(nsaddr_t id, bool error)
{
    aodv_rt_entry *rt, *rtn;
    Packet *rerr = Packet::alloc();
    struct hdr_aodv_error *re = HDR_AODV_ERROR(rerr);

    #ifdef DEBUG
    fprintf(stderr, "%s: multipath version\n", __FUNCTION__);
    #endif // DEBUG

    re->DestCount = 0;
    for(rt = rtable.head(); rt; rt = rtn)
    { // for each rt entry
        AODV_Path* path;
        rtn = rt->rt_link.le_next;
        if ((rt->rt_flags == RTF_UP) && (path=rt->path_lookup(id)) )
        {
            assert((rt->rt_seqno % 2) == 0);

            // [scheduling] ——
            #ifdef SA_CHANGE
            #ifdef AODV_INFO
            int num_path = rt->rt_num_paths_ -1;
            dumpPath(rt, path, -1, num_path);
            #endif
            #endif
            // [scheduling] —— end

            rt->path_delete(id);
            if (rt->path_empty())
            {
                rt->rt_seqno++;
                rt->rt_seqno = max(rt->rt_seqno, rt->rt_highest_seqno_heard);
                // CHANGE
                if (rt->rt_error)
                {
                    re->unreachable_dst[re->DestCount] = rt->rt_dst;
                    re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
                    #ifdef DEBUG
                    fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\n", __FUNCTION__, NOW,
                            index, re->unreachable_dst[re->DestCount],
                            re->unreachable_dst_seqno[re->DestCount], id);
                    #endif // DEBUG
                    re->DestCount += 1;
                    rt->rt_error = false;
                }
                // CHANGE
                rt_down(rt);
            }
        }
    }
```

```
    if ( (re->DestCount > 0) && (error) )
    {
        #ifdef DEBUG
        fprintf(stdout, "%s(%f): %d\tsending RERR...\n", __FUNCTION__, NOW, index);
        #endif // DEBUG
        sendError(rerr, false);
    }
    else
    {
        Packet::free(rerr);
    }
}

void AODV::rt_down(aodv_rt_entry *rt)
{
    /*
    * Make sure that you don't "down" a route more than once.
    */

    #ifdef DEBUG
    fprintf(stderr, "%s: multipath version\n", __FUNCTION__);
    #endif // DEBUG

    if(rt->rt_flags == RTF_DOWN)
    {
        return;
    }

    assert (rt->rt_seqno % 2); // is the seqno odd?
    rt->rt_flags = RTF_DOWN;
    rt->rt_advertised_hops = INFINITY;
    // [scheduling] ---
    #ifdef SA_CHANGE
    // print the paths before they are removed
    #ifdef AODV_INFO
    AODV_Path *path = rt->rt_path_list.lh_first;
    int num_path = rt->rt_num_paths_;
    for(; path; path = path->path_link.le_next)
    {
        num_path--;
        dumpPath(rt, path, -1, num_path);
    }
    #endif
    #endif
    // [scheduling] --- end
    rt->path_delete();
    rt->rt_expire = 0;
} /* rt_down function */

void AODV::recvError(Packet *p)
{
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_aodv_error *re = HDR_AODV_ERROR(p);
    aodv_rt_entry *rt;
    u_int8_t i;
    Packet *rerr = Packet::alloc();
    struct hdr_aodv_error *nre = HDR_AODV_ERROR(rerr);

    #ifdef DEBUG
    fprintf(stderr, "%s: multipath version\n", __FUNCTION__);
    #endif // DEBUG


    nre->DestCount = 0;
    // For each unreachable destination
    for (i=0; i<re->DestCount; i++)
```

```
    {
        AODV_Path* path;
        /* Get route entry for unreachable dest */
        rt = rtable.rt_lookup(re->unreachable_dst[i]);
        /* If route entry exists, route is up, a path to the unreachable
        destination exists through the neigbor from which RERR was
        received, and my sequence number is not more recent - delete
        path and add it to the RERR message I will send. */
        if ( rt && (rt->rt_flags == RTF_UP) &&
            (path = rt->path_lookup(ih->saddr())) &&
            (rt->rt_seqno <= re->unreachable_dst_seqno[i]) )
        {
            assert((rt->rt_seqno % 2) == 0); // is the seqno even?
            // [scheduling] ---
            #ifdef SA_CHANGE
            #ifdef AODV_INFO
            int num_path = rt->rt_num_paths_ -1;
            dumpPath(rt, path, -1, num_path);
            #endif
            #endif
            // [scheduling] --- end

            rt->path_delete(ih->saddr());
            rt->rt_highest_seqno_heard = max(rt->rt_highest_seqno_heard,
                                re->unreachable_dst_seqno[i]);
            if (rt->path_empty())
            {
                rt->rt_seqno = rt->rt_highest_seqno_heard;
                rt_down(rt);
                // CHANGE
                if (rt->rt_error)
                {
                    nre->unreachable_dst[nre->DestCount] = rt->rt_dst;
                    nre->unreachable_dst_seqno[nre->DestCount] = rt->rt_seqno;
                    nre->DestCount += 1;
                    rt->rt_error = false;
                }
                // CHANGE
            }
        }
    }
    /* Broadcast RERR if any broken paths were found. */
    if (nre->DestCount > 0)
    {
        #ifdef DEBUG
        fprintf(stderr, "%s(%f): %d\t sending RERR...\n", __FUNCTION__, NOW, index);
        #endif // DEBUG
        sendError(rerr);
    }
    else
    {
        Packet::free(rerr);
    }

    Packet::free(p);
}
#endif //AOMDV
```

Listing A.2: aodv.cc

A.1.3. aodv_rtable.h

```
/* -*-  Mode:C++; c-basic-offset:4; tab-width:4; indent-tabs-mode:t -*- */
/*
Copyright (c) 1997, 1998 Carnegie Mellon University. All Rights
Reserved.

Permission to use, copy, modify, and distribute this software and its documentation is
hereby granted (including for commercial or for-profit use), provided that both the
copyright notice and this permission notice appear in all copies of the software,
derivative works, or modified versions, and any portions thereof, and that both notices
appear in supporting documentation, and that credit is given to Carnegie Mellon University
in all publications reporting on direct or indirect use of this code or its derivatives.

ALL CODE, SOFTWARE, PROTOCOLS, AND ARCHITECTURES DEVELOPED BY THE CMU MONARCH PROJECT ARE
EXPERIMENTAL AND ARE KNOWN TO HAVE BUGS, SOME OF WHICH MAY HAVE SERIOUS CONSEQUENCES.
CARNEGIE MELLON PROVIDES THIS SOFTWARE OR OTHER INTELLECTUAL PROPERTY IN ITS ''AS IS''
CONDITION, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE OR INTELLECTUAL PROPERTY, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.

Carnegie Mellon encourages (but does not require) users of this software or intellectual
property to return any improvements or extensions that they make, and to grant Carnegie
Mellon the rights to redistribute these changes without encumbrance.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and
Mahesh Marina, University of Cincinnati. The work was partially done in Sun Microsystems.

#-----------------------------------------------------------------
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#
# Date:     July 2006
#
# The code has been modiefied by R. Goenner and D.Schatzmann,
# modifications are indicated with the tag // [scheduling] ---
# and they can be activated by using #define SA_CHANGE
#-----------------------------------------------------------------

*/


#ifndef __aodv_rtable_h__
#define __aodv_rtable_h__

#include <assert.h>
#include <sys/types.h>
#include <config.h>
#include <lib/bsd-list.h>
#include <scheduler.h>

#define CURRENT_TIME Scheduler::instance().clock()
#define INFINITY2 0xff

// [scheduling] ---
// we need a timeout for each path, set it the same as the route timeout
#define ACTIVE_PATH_TIMEOUT 10  // 10 seconds
// [scheduling] --- end
```

```
/*
 AODV Neighbor Cache Entry
*/
class AODV_Neighbor
{
    friend class AODV;
    friend class aodv_rt_entry;
    public:
    AODV_Neighbor(u_int32_t a) { nb_addr = a; }

    protected:
    LIST_ENTRY(AODV_Neighbor) nb_link;
    nsaddr_t nb_addr;
    double nb_expire; // ALLOWED_HELLO_LOSS * HELLO_INTERVAL
};

LIST_HEAD(aodv_ncache, AODV_Neighbor);

#ifdef AOMDV
/*
 AODV Path list data structure
*/
class AODV_Path
{
    friend class AODV;
    friend class aodv_rt_entry;
    public:
        AODV_Path(nsaddr_t nh, u_int16_t h, double expire_time, nsaddr_t lh=0)
        {
            nexthop     = nh;
            hopcount    = h;
            expire      = expire_time;
            ts          = Scheduler::instance().clock();
            lasthop     = lh;
            // [scheduling] ---
            #ifdef SA_CHANGE
            rtt         = 1000.0;
            nr_neigh    = INT_MAX;
            #endif
            // [scheduling] --- end
            // CHANGE
            error = false;
            // CHANGE
        }
        void printPath()
        {
            printf(" %6d %6d %6d\n", nexthop, hopcount, lasthop);
        }
        void printPaths()
        {
            AODV_Path *p = this;
            for (; p; p = p->path_link.le_next)
            {
                p->printPath();
            }
        }

    protected:
        LIST_ENTRY(AODV_Path) path_link;
        nsaddr_t    nexthop;    // nexthop address
        u_int16_t   hopcount;   // hopcount through this nexthop
        double      expire;     // expiration timeout
        double      ts;         // time when we saw this nexthop
        nsaddr_t    lasthop;    // lasthop address
        // CHANGE
        bool        error;
```

```
        // CHANGE
        // [scheduling] ——
        #ifdef SA_CHANGE
        double      rtt;
        int         nr_neigh;
        double      ping_sent;
        #endif
        // [scheduling] —— end
};

LIST_HEAD(aodv_paths, AODV_Path);
#endif // AOMDV

/*
 AODV Precursor list data structure
*/
class AODV_Precursor
{
    friend class AODV;
    friend class aodv_rt_entry;

    public:
        AODV_Precursor(u_int32_t a) { pc_addr = a; }

    protected:
        LIST_ENTRY(AODV_Precursor) pc_link;
        nsaddr_t pc_addr;       // precursor address
};

LIST_HEAD(aodv_precursors, AODV_Precursor);


/*
 Route Table Entry
*/

class aodv_rt_entry
{
        friend class aodv_rtable;
        friend class AODV;
        friend class LocalRepairTimer;
    public:
        aodv_rt_entry();
        ~aodv_rt_entry();

        void nb_insert(nsaddr_t id);
        AODV_Neighbor* nb_lookup(nsaddr_t id);

        #ifdef AOMDV
        AODV_Path*  path_insert(nsaddr_t nexthop, u_int16_t hopcount,
                        double expire_time, nsaddr_t lasthop=0);

        AODV_Path*  path_lookup(nsaddr_t id);   // lookup path by nexthop

        AODV_Path*  disjoint_path_lookup(nsaddr_t nexthop, nsaddr_t lasthop);
        bool        new_disjoint_path(nsaddr_t nexthop, nsaddr_t lasthop);

        AODV_Path*  path_lookup_lasthop(nsaddr_t id);   // lookup path by lasthop
        void        path_delete(nsaddr_t id);           // delete path by nexthop
        void        path_delete(void);                  // delete all paths
        void        path_delete_longest(void);          // delete longest path
        bool        path_empty(void);                   // path list empty?
        #ifdef SA_CHANGE
        AODV_Path*  path_find(nsaddr_t src, nsaddr_t index);// find a path to dest
        #else
        AODV_Path*  path_find();                        // find a path to dest
        #endif
```

```
    AODV_Path*  path_findMinHop(void);              // find the shortest path
    u_int16_t   path_get_max_hopcount(void);
    u_int16_t   path_get_min_hopcount(void);
    double      path_get_max_expiration_time(void);
    void        path_purge(void);
    #endif // AOMDV

    void        pc_insert(nsaddr_t id);
    AODV_Precursor* pc_lookup(nsaddr_t id);
    void        pc_delete(nsaddr_t id);
    void        pc_delete(void);
    bool        pc_empty(void);

    double      rt_req_timeout;                     // when I can send another req
    u_int8_t    rt_req_cnt;                         // number of route req

    int     aomdv_max_paths_;

protected:
    LIST_ENTRY(aodv_rt_entry) rt_link;

    nsaddr_t    rt_dst;
    u_int32_t   rt_seqno;
    /* u_int8_t rt_interface; */

    // [scheduling] ---
    #ifdef SA_CHANGE
    #ifdef AOMDV
    int         sched_type;
    void        sched_calc_weights();           // calculate the weigthing
    AODV_Path*  sched_get_path();               // select a path
    void        path_ping();
    int*        weights;
    void        path_dumptable() ;
    nsaddr_t    tx_rx_id;                       // store the dest
    #endif
    #endif
    // [scheduling] --- end
    #ifndef AOMDV
    u_int16_t   rt_hops;                        // hop count
    #else
    u_int16_t   rt_advertised_hops;             // advertised hop count
    #endif // AOMDV

    int         rt_last_hop_count;              // last valid hop count

    #ifndef AOMDV
    nsaddr_t    rt_nexthop;                     // next hop IP address
    #else
    aodv_paths  rt_path_list;                   // list of paths
    u_int32_t   rt_highest_seqno_heard;
    int         rt_num_paths_;
    #endif // AOMDV

    // CHANGE
    bool        rt_error;
    // CHANGE

    /* list of precursors */
    aodv_precursors rt_pclist;
    double      rt_expire;                      // when entry expires
    u_int8_t    rt_flags;

    #define RTF_DOWN 0
    #define RTF_UP 1
    #define RTF_IN_REPAIR 2
```

```
        #define MAX_HISTORY 3
        double      rt_disc_latency[MAX_HISTORY];
        char        hist_indx;
        int         rt_req_last_ttl;                // last ttl value used

        /*
        * a list of neighbors that are using this route.
        */
        aodv_ncache     rt_nblist;
};


/*
 The Routing Table
*/

class aodv_rtable
{
    #ifdef SA_CHANGE
    friend class AODV;  // [scheduling]
    #endif
    public:
        aodv_rtable() { LIST_INIT(&rthead); }

        aodv_rt_entry*  head() { return rthead.lh_first; }

        aodv_rt_entry*  rt_add(nsaddr_t id);
        void            rt_delete(nsaddr_t id);
        aodv_rt_entry*  rt_lookup(nsaddr_t id);
        void            rt_dumptable();
        bool            rt_has_active_route();
        // [scheduling] ---
        #ifdef SA_CHANGE
        int             sched_type;
        #endif
        // [scheduling] --- end
        int     aomdv_max_paths_;
    private:
        LIST_HEAD(aodv_rthead, aodv_rt_entry) rthead;

};

#endif /* _aodv__rtable_h__ */
```

Listing A.3: aodv_rtable.h

A.1.4. aodv_rtable.cc

```
/* -*-  Mode:C++; c-basic-offset:4; tab-width:4; indent-tabs-mode:t -*- */
/*
Copyright (c) 1997, 1998 Carnegie Mellon University.  All Rights
Reserved.

Permission to use, copy, modify, and distribute this
software and its documentation is hereby granted (including for commercial or for-profit
use), provided that both the copyright notice and this permission notice appear in all
copies of the software, derivative works, or modified versions, and any portions thereof,
and that both notices appear in supporting documentation, and that credit is given to
Carnegie Mellon University in all publications reporting on direct or indirect use of this
code or its derivatives.

ALL CODE, SOFTWARE, PROTOCOLS, AND ARCHITECTURES DEVELOPED BY THE CMU MONARCH PROJECT ARE
EXPERIMENTAL AND ARE KNOWN TO HAVE BUGS, SOME OF WHICH MAY HAVE SERIOUS CONSEQUENCES.
CARNEGIE MELLON PROVIDES THIS SOFTWARE OR OTHER INTELLECTUAL PROPERTY IN ITS ''AS IS''
CONDITION, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE OR INTELLECTUAL PROPERTY, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.

Carnegie Mellon encourages (but does not require) users of this software or intellectual
property to return any improvements or extensions that they make, and to grant Carnegie
Mellon the rights to redistribute these changes without encumbrance.
The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and
Mahesh Marina, University of Cincinnati. The work was partially done in Sun Microsystems.

#-----------------------------------------------------------------------
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#
# Date:     July 2006
#
# The code has been modiefied by R. Goenner and D.Schatzmann,
# modifications are indicated with the tag // [scheduling] ---
# and they can be activated by using #define SA_CHANGE
#-----------------------------------------------------------------------

*/

#include <iostream>
#include <random.h>
#include <aodv/aodv_rtable.h>
//#include <cmu/aodv/aodv.h>

#define NOW Scheduler::instance().clock()

/*
  The Routing Table
*/

aodv_rt_entry::aodv_rt_entry()
{
    int i;

    rt_req_timeout = 0.0;
    rt_req_cnt = 0;
```

```
    rt_dst = 0;
    rt_seqno = 0;
    rt_last_hop_count = INFINITY2;

    #ifndef AOMDV
    rt_hops = INFINITY2;
    rt_nexthop = 0;
    #else // AOMDV
    rt_advertised_hops = INFINITY2;
    LIST_INIT(&rt_path_list);
    rt_highest_seqno_heard = 0;
    rt_num_paths_ = 0;
    #endif // AOMDV

    // CHANGE
    rt_error = false;
    // CHANGE

    LIST_INIT(&rt_pclist);
    rt_expire = 0.0;
    rt_flags = RTF_DOWN;


    for (i=0; i < MAX_HISTORY; i++)
    {
        rt_disc_latency[i] = 0.0;
    }
    hist_indx = 0;
    rt_req_last_ttl = 0;

    LIST_INIT(&rt_nblist);

    // [scheduling] ——
    #ifdef SA_CHANGE
    tx_rx_id = -INT_MAX;                // init the destination
    #endif
    // [scheduling] —— end

};


aodv_rt_entry::~aodv_rt_entry()
{
    AODV_Neighbor *nb;

    while((nb = rt_nblist.lh_first))
    {
        LIST_REMOVE(nb, nb_link);
        delete nb;
    }

    #ifdef AOMDV
    AODV_Path *path;

    while((path = rt_path_list.lh_first))
    {
        LIST_REMOVE(path, path_link);
        delete path;
    }
    #endif // AOMDV

    AODV_Precursor *pc;

    while((pc = rt_pclist.lh_first))
    {
        LIST_REMOVE(pc, pc_link);
```

```
        delete pc;
    }

}


void aodv_rt_entry::nb_insert(nsaddr_t id)
{
    AODV_Neighbor *nb = new AODV_Neighbor(id);

    assert(nb);
    nb->nb_expire = 0;
    LIST_INSERT_HEAD(&rt_nblist, nb, nb_link);

}


AODV_Neighbor* aodv_rt_entry::nb_lookup(nsaddr_t id)
{
    AODV_Neighbor *nb = rt_nblist.lh_first;

    for(; nb; nb = nb->nb_link.le_next)
    {
        if(nb->nb_addr == id)
        break;
    }
    return nb;

}

#ifdef AOMDV
AODV_Path* aodv_rt_entry::path_insert(nsaddr_t nexthop, u_int16_t hopcount,
    double expire_time, nsaddr_t lasthop)
{
    AODV_Path *path = new AODV_Path(nexthop, hopcount, expire_time, lasthop);

    assert(path);
    #ifdef DEBUG
    fprintf(stderr, "%s: (%d\t%d)\n", __FUNCTION__, path->nexthop, path->hopcount);
    #endif // DEBUG

    /*
    * Insert path at the end of the list
    */
    AODV_Path *p = rt_path_list.lh_first;
    if (p)
    {
        for(; p->path_link.le_next; p = p->path_link.le_next)
            /* Do nothing */;
        LIST_INSERT_AFTER(p, path, path_link);
    }
    else
    {
        LIST_INSERT_HEAD(&rt_path_list, path, path_link);
    }
    rt_num_paths_ += 1;
    // [scheduling] ---
    #ifdef SA_CHANGE
    sched_calc_weights();
    #endif
    // [scheduling] --- end
    return path;

}

AODV_Path* aodv_rt_entry::path_lookup(nsaddr_t id)
{
```

```
    AODV_Path *path = rt_path_list.lh_first;

    for(; path; path = path->path_link.le_next)
    {
        if (path->nexthop == id)
            return path;
    }
    return NULL;

}

AODV_Path* aodv_rt_entry::disjoint_path_lookup(nsaddr_t nexthop, nsaddr_t lasthop)
{
    AODV_Path *path = rt_path_list.lh_first;

    for(; path; path = path->path_link.le_next)
    {
        if ( (path->nexthop == nexthop) && (path->lasthop == lasthop) )
            return path;
    }
    return NULL;

}

/* Returns true if no path exists (for this route entry) which has 'nexthop' as next hop
 * or 'lasthop' as last hop.*/
bool aodv_rt_entry::new_disjoint_path(nsaddr_t nexthop, nsaddr_t lasthop)
{
    AODV_Path *path = rt_path_list.lh_first;

    for(; path; path = path->path_link.le_next)
    {
        if ( (path->nexthop == nexthop) || (path->lasthop == lasthop) )
            return false;
    }
    return true;

}


AODV_Path* aodv_rt_entry::path_lookup_lasthop(nsaddr_t id)
{
    AODV_Path *path = rt_path_list.lh_first;

    for(; path; path = path->path_link.le_next)
    {
        if (path->lasthop == id)
            return path;
    }
    return NULL;

}


void aodv_rt_entry::path_delete(nsaddr_t id)
{
    AODV_Path *path = rt_path_list.lh_first;
    for(; path; path = path->path_link.le_next)
    {
        if(path->nexthop == id)
        {
            LIST_REMOVE(path,path_link);
            delete path;
            rt_num_paths_ -= 1;
            break;
        }
    }
```

```
    // [scheduling] ---
    #ifdef SA_CHANGE
    sched_calc_weights();
    #endif
    // [scheduling] --- end
}

void aodv_rt_entry::path_delete(void)
{
    AODV_Path *path;
    while((path = rt_path_list.lh_first))
    {
        LIST_REMOVE(path, path_link);
        delete path;
    }
    rt_num_paths_ = 0;
    // [scheduling] ---
    #ifdef SA_CHANGE
    sched_calc_weights();
    #endif
    // [scheduling] --- end
}

void aodv_rt_entry::path_delete_longest(void)
{
    AODV_Path *p = rt_path_list.lh_first;
    AODV_Path *path = NULL;
    u_int16_t max_hopcount = 0;

    for(; p; p = p->path_link.le_next)
    {
        if(p->hopcount > max_hopcount)
        {
            assert (p->hopcount != INFINITY2);
            path = p;
            max_hopcount = p->hopcount;
        }
    }

    if (path)
    {
        LIST_REMOVE(path, path_link);
        delete path;
        rt_num_paths_ -= 1;
    }
    // [scheduling] ---
    #ifdef SA_CHANGE
    sched_calc_weights();
    #endif
    // [scheduling] -- end

}

bool aodv_rt_entry::path_empty(void)
{
    AODV_Path *path;

    if ((path = rt_path_list.lh_first))
    {
        assert (rt_num_paths_ > 0);
        return false;
    }
    else
    {
        assert (rt_num_paths_ == 0);
        return true;
    }
```

```
}

AODV_Path* aodv_rt_entry::path_findMinHop(void)
{
    AODV_Path *p = rt_path_list.lh_first;
    AODV_Path *path = NULL;
    u_int16_t min_hopcount = 0xffff;

    for (; p; p = p->path_link.le_next)
    {
        if (p->hopcount < min_hopcount)
        {
            path = p;
            min_hopcount = p->hopcount;
        }
    }

    return path;
}

// [scheduling] ---
#ifdef SA_CHANGE
AODV_Path* aodv_rt_entry::path_find(nsaddr_t src, nsaddr_t index)
#else
AODV_Path* aodv_rt_entry::path_find()
#endif
{
    AODV_Path *p;
    // if  this is the sender
    #ifdef SA_CHANGE
    if (index == src)
        p = sched_get_path();
    // do not schedule in intermediate nodes
    else
    #endif
        p = rt_path_list.lh_first;
    // the scheduler returns the first path of the list but this entry can change,
    // as the information (e.g rtt) depend on the path it would make sense to store
    // the id of the next hop that was used. If this path does no longer exist a new
    // ping would be required and this should be indicated to the source.
    #ifdef SA_CHANGE
    // we have to reset the path timeout
    p->expire = NOW + ACTIVE_PATH_TIMEOUT;
    #endif
    return p;
}

#ifdef SA_CHANGE
void aodv_rt_entry::sched_calc_weights()
{ // calculate the weighting values according to a certain criteria
    AODV_Path *p        = rt_path_list.lh_first;
    int total_value     = 0;    // use this to find the sum of the values
    int prev_value      = 0;
    int total_diff      = 0;
    int num_paths       = 0;
    int total_hopcount  = 0;

    for (; p; p = p->path_link.le_next)
    {
        num_paths++;
        total_hopcount += p->hopcount;
    }

    switch(sched_type)
    {
        case 0:
        { // AOMDV
```

```
        p = rt_path_list.lh_first; // reset

        int min = INT_MAX;
        int idx = 0;
        // set the cumulated hopcounts in the weights array
        for (int l = 0; l< num_paths; l++)
        {
            if (p->hopcount < min)
            {
                min = p->hopcount;
                idx = l;
            }
            p = p->path_link.le_next;
        }
        for (int m = 0; m < num_paths; m++)
        {
            if (m >= idx)
            {
                weights[m]=INT_MAX;
            }
            else
            {
                weights[m] = 0;
            }
        }
    }
    break;
    //--------------
    case 1:
    { // RR
        total_value = num_paths;
        for (int i=0;i< num_paths;i++)
        {
            weights[i]=(INT_MAX*1)/total_value + prev_value;
            prev_value = weights[i];
        }
    }
    break;
    //--------------
    case 2:
    { // WRR, the shortest path is used most often
        p = rt_path_list.lh_first; // reset
        total_diff = (num_paths -1)*total_hopcount;
        for (int i=0;i< num_paths;i++)
        {
            if (total_diff == 0 )
            {
                if (num_paths == 1)
                {
                    weights[i] = INT_MAX;
                }
                else cout << "error, check WRR in aodv_rtable" ;

            }
            else
            {
                weights[i] = int(INT_MAX*(double((total_hopcount -
                        p->hopcount)/double(total_diff))) + prev_value);
                prev_value = weights[i];
                p = p->path_link.le_next;
            }
        }
    }
    break;
    //--------------
    case 3:
    {// SN, select N path and WRR
```

```
    /*
    if you want to use only some of the path, here all path
    with a hopcount that is larger than 20% of the mean are not used
    */

    double mean;
    mean = total_hopcount/double(num_paths);

    int threshold;
    threshold = int(1.2 * mean);

    p = rt_path_list.lh_first; // reset
    total_value = 0;

    int prev_value;
    prev_value = 0;

    // set the cumulated hopcounts in the weights array
    for (int l = 0; l < num_paths; l++)
    {
        if (p->hopcount <= threshold)
        {
            total_value += p->hopcount;
            weights[l] = p->hopcount + prev_value;
            prev_value = weights[l];
        }
        else
        {
            weights[l] = 0;
        }
        p = p->path_link.le_next;

    }
    // calculate the weights (div by 'total_value' and mult by INT_MAX
    for (int m = 0; m < num_paths; m++)
    {
        if (total_value != 0)
        {
            weights[m] = int(double((weights[m]/ double(total_value)) * INT_MAX));
        }
        else
        {
            weights[m] = INT_MAX;
        }
    }
}
break;
//-------------
case 4:
{ // WRR with the rtt
    p = rt_path_list.lh_first; // reset
    double total_rtt = 0.0;
    for (int i=0; i<num_paths; i++)
    {
        total_rtt += p->rtt;
        p = p->path_link.le_next;
    }
    double total_rtt_diff = (num_paths - 1) * total_rtt;
    p = rt_path_list.lh_first;
    for (int j = 0; j < num_paths; j++)
    {
        if (total_diff == 0 && num_paths == 1)
        {
            weights[j] = INT_MAX;
        }
        else
        {
```

```
                weights[j] = int(INT_MAX*(double((total_rtt -
                    p->rtt)/total_rtt_diff)) + prev_value);
                prev_value = weights[j];
                p = p->path_link.le_next;
            }
        }

    }
    break;
    //--------------
    case 5:
    { // select the path with lowest number of neighbours
        p = rt_path_list.lh_first; // reset
        int min_neigh = INT_MAX;
        for (int i = 0; i < num_paths; i++)
        {
            if (p->nr_neigh < min_neigh)
                min_neigh = p->nr_neigh;
            p = p->path_link.le_next;
        }

        p = rt_path_list.lh_first;
        int flag_found = 0;
        for (int j = 0; j < num_paths; j++)
        {
            if (flag_found == 0)
            {
                if (p->nr_neigh = min_neigh)
                {
                    weights[j] = INT_MAX;
                    flag_found = 1;
                }
                else
                {
                    weights[j] = 0;
                }
            }
            else
            {
                weights[j] = INT_MAX;
            }
            p = p->path_link.le_next;
        }

    }
    break;
    //--------------
    default:
        cout << "This scheduler does not exist! \n";
    break;
    //--------------
}
//--------------
#ifdef SA_DEBUG
path_dumptable();
if (num_paths > 0)
{
    cout <<"weights  |";
    for (int i = 0; i < num_paths; i++)
    {
        cout << " " << weights[i] << " |";
    }
    cout << "\n";

    cout <<"hopcount |";
    p = rt_path_list.lh_first;
    for (int i = 0; i < num_paths; i++)
```

```
        {
            cout << " " << p->hopcount << " |";
            p = p->path_link.le_next;
        }
        cout << "\n";

        cout <<"rtt      |";
        p = rt_path_list.lh_first;
        for (int i = 0; i < num_paths; i++)
        {
            cout << " " << p->rtt << " |";
            p = p->path_link.le_next;
        }
        cout << "\n";

        cout <<"next hop |";
        p = rt_path_list.lh_first;
        for (int i = 0; i < num_paths; i++)
        {
            cout << " " << p->nexthop << " |";
            p = p->path_link.le_next;
        }
        cout << "\n";

        cout <<"neighs |";
        p = rt_path_list.lh_first;
        for (int i = 0; i < num_paths; i++)
        {
            cout << " " << p->nr_neigh << " |";
            p = p->path_link.le_next;
        }
        cout << "\n";
    }
    #endif
    //---------------

}


AODV_Path* aodv_rt_entry::sched_get_path()
{ // roll the dice and return the path
    AODV_Path *p = rt_path_list.lh_first;
    int num_paths = 0;
    for (; p; p = p->path_link.le_next) //count the number of paths
    {
        num_paths++;
    }

    // create values in the range of 0 to INT_MAX-1
    int random = int( Random::uniform(INT_MAX))%INT_MAX;
    p = rt_path_list.lh_first;      // reset p
    int i = 0;
    while (i < num_paths)
    {
        if (weights[i] >= random)
        {
            break;
        }
        else
        {
            i++;
            p = p->path_link.le_next;
        }
    }
    return p;
}
// [scheduling] --- end
```

```
#endif

u_int16_t aodv_rt_entry::path_get_max_hopcount(void)
{
    AODV_Path *path = rt_path_list.lh_first;
    u_int16_t max_hopcount = 0;

    for(; path; path = path->path_link.le_next)
    {
        if(path->hopcount > max_hopcount)
        {
            max_hopcount = path->hopcount;
        }
    }
    if (max_hopcount == 0) return INFINITY2;
    else return max_hopcount;
}

u_int16_t aodv_rt_entry::path_get_min_hopcount(void)
{
    AODV_Path *path = rt_path_list.lh_first;
    u_int16_t min_hopcount = INFINITY2;

    for(; path; path = path->path_link.le_next)
    {
        if(path->hopcount < min_hopcount)
        {
            min_hopcount = path->hopcount;
        }
    }
    return min_hopcount;
}

double aodv_rt_entry::path_get_max_expiration_time(void)
{
    AODV_Path *path = rt_path_list.lh_first;
    double max_expire_time = 0;

    for(; path; path = path->path_link.le_next)
    {
        if(path->expire > max_expire_time)
        {
            max_expire_time = path->expire;
        }
    }
    return max_expire_time;
}

void aodv_rt_entry::path_purge(void)
{
    double now = Scheduler::instance().clock();
    bool cond;

    do
    {
        AODV_Path *path = rt_path_list.lh_first;
        cond = false;
        for(; path; path = path->path_link.le_next)
        {
            if(path->expire < now)
            {
                cond = true;
                LIST_REMOVE(path, path_link);
                delete path;
                rt_num_paths_ -= 1;
                // [scheduling] ---
                #ifdef SA_CHANGE
```

```
                sched_calc_weights();
                #endif
                // [scheduling] ---- end
                break;
            }
        }
    } while (cond);

}

#endif // AOMDV

void aodv_rt_entry::pc_insert(nsaddr_t id)
{
    if (pc_lookup(id) == NULL)
    {
        AODV_Precursor *pc = new AODV_Precursor(id);

        assert(pc);
        LIST_INSERT_HEAD(&rt_pclist, pc, pc_link);
    }
}


AODV_Precursor* aodv_rt_entry::pc_lookup(nsaddr_t id)
{
    AODV_Precursor *pc = rt_pclist.lh_first;

    for(; pc; pc = pc->pc_link.le_next)
    {
        if(pc->pc_addr == id)
            return pc;
    }
    return NULL;
}

void aodv_rt_entry::pc_delete(nsaddr_t id)
{
    AODV_Precursor *pc = rt_pclist.lh_first;

    for(; pc; pc = pc->pc_link.le_next)
    {
        if(pc->pc_addr == id)
        {
            LIST_REMOVE(pc,pc_link);
            delete pc;
            break;
        }
    }
}

void aodv_rt_entry::pc_delete(void)
{
    AODV_Precursor *pc;

    while((pc = rt_pclist.lh_first))
    {
        LIST_REMOVE(pc, pc_link);
        delete pc;
    }
}

bool aodv_rt_entry::pc_empty(void)
{
    AODV_Precursor *pc;

    if ((pc = rt_pclist.lh_first)) return false;
```

```
    else return true;
}

/*
  The Routing Table
*/

aodv_rt_entry* aodv_rtable::rt_lookup(nsaddr_t id)
{
    aodv_rt_entry *rt = rthead.lh_first;

    for(; rt; rt = rt->rt_link.le_next)
    {
        if(rt->rt_dst == id)
            break;
    }
    return rt;
}

void aodv_rtable::rt_delete(nsaddr_t id)
{
    aodv_rt_entry *rt = rt_lookup(id);

    if(rt)
    {
        LIST_REMOVE(rt, rt_link);
        delete rt;
    }
}

aodv_rt_entry* aodv_rtable::rt_add(nsaddr_t id)
{
    aodv_rt_entry *rt;

    assert(rt_lookup(id) == 0);
    rt = new aodv_rt_entry;
    assert(rt);
    rt->rt_dst = id;
    LIST_INSERT_HEAD(&rthead, rt, rt_link);
    // [scheduling] ---
    #ifdef SA_CHANGE
    rt->sched_type = sched_type;
    rt->aomdv_max_paths_ = aomdv_max_paths_;
    rt->weights = new int[aomdv_max_paths_];
    rt->sched_calc_weights();
    #endif
    // [scheduling] --- end
    return rt;
}

void aodv_rtable::rt_dumptable()
{
    aodv_rt_entry *rt = rthead.lh_first;
    printf("----------\n");
    while(rt != 0)
    {
        printf("%6s  %6s  ", "Dest", "Seq#");
        #ifndef AOMDV
        printf("%6s  %6s\n", "Hops", "Nxthop");
        #else
        printf("%6s  %6s  %6s  %6s\n", "Advhop", "Nxthop", "Hopcnt", "Lsthop");
        #endif // AOMDV

        printf("%6d  %6d  ", rt->rt_dst, rt->rt_seqno);
        #ifndef AOMDV
        printf("%6d %6d\n", rt->rt_hops, rt->rt_nexthop);
        #else
```

```
        printf("%6d\n", rt->rt_advertised_hops);
        /* Print path list for this route entry. */
        AODV_Path *paths = rt->rt_path_list.lh_first;
        paths->printPaths();
        #endif // AOMDV
        printf("\n");
        rt = rt->rt_link.le_next;
    }
}

bool aodv_rtable::rt_has_active_route()
{
    /* Go through list of route entries to see if there exists
    any valid route entry */
    aodv_rt_entry *rt = rthead.lh_first;
    while(rt != 0)
    {
        if (rt->rt_flags == RTF_UP)
            return true;
        rt = rt->rt_link.le_next;
    }
    return false;
}
```

Listing A.4: aodv_rtable.cc

A.1.5. aodv_packet.h

```
/* -*-  Mode:C++; c-basic-offset:4; tab-width:4; indent-tabs-mode:t -*- */
/*
Copyright (c) 1997, 1998 Carnegie Mellon University. All Rights
Reserved.

Permission to use, copy, modify, and distribute this software and its documentation is
hereby granted (including for commercial or for-profit use), provided that both the
copyright notice and this permission notice appear in all copies of the software,
derivative works, or modified versions, and any portions thereof, and that both notices
appear in supporting documentation, and that credit is given to Carnegie Mellon University
in all publications reporting on direct or indirect use of this code or its derivatives.

ALL CODE, SOFTWARE, PROTOCOLS, AND ARCHITECTURES DEVELOPED BY THE CMU MONARCH PROJECT ARE
EXPERIMENTAL AND ARE KNOWN TO HAVE BUGS, SOME OF WHICH MAY HAVE SERIOUS CONSEQUENCES.
CARNEGIE MELLON PROVIDES THIS SOFTWARE OR OTHER INTELLECTUAL PROPERTY IN ITS ''AS IS''
CONDITION, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OF THIS SOFTWARE OR INTELLECTUAL PROPERTY, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

Carnegie Mellon encourages (but does not require) users of this software or intellectual
property to return any improvements or extensions that they make, and to grant Carnegie
Mellon the rights to redistribute these changes without encumbrance.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and
Mahesh Marina, University of Cincinnati. The work was partially done in Sun Microsystems.

#-----------------------------------------------------------------
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#
# Date:     July 2006
#
# The code has been modiefied by R. Goenner and D.Schatzmann,
# modifications are indicated with the tag // [scheduling] ---
# and they can be activated by using #define SA_CHANGE
#-----------------------------------------------------------------

*/


#ifndef __aodv_packet_h__
#define __aodv_packet_h__

#define AODV_MAX_ERRORS 100


/* ======================================================================
 Packet Formats...
 ====================================================================== */
#define AODVTYPE_HELLO 0x01
#define AODVTYPE_RREQ 0x02
#define AODVTYPE_RREP 0x04
#define AODVTYPE_RERR 0x08
#define AODVTYPE_RREP_ACK 0x10
#define AODVTYPE_PING 0x06
#define AODVTYPE_INFO 0x07
```

```
/*
 * AODV Routing Protocol Header Macros
 */
#define HDR_AODV(p)          ((struct hdr_aodv*)hdr_aodv::access(p))
#define HDR_AODV_REQUEST(p) ((struct hdr_aodv_request*)hdr_aodv::access(p))
#define HDR_AODV_REPLY(p)   ((struct hdr_aodv_reply*)hdr_aodv::access(p))
// [scheduling] ——
#ifdef SA_CHANGE
#define HDR_AODV_PING(p)    ((struct hdr_aodv_ping*)hdr_aodv::access(p))
#define HDR_AODV_INFO(p)    ((struct hdr_aodv_info*)hdr_aodv::access(p))
#endif
// [scheduling] —— end
#define HDR_AODV_ERROR(p)   ((struct hdr_aodv_error*)hdr_aodv::access(p))
#define HDR_AODV_RREP_ACK(p)((struct hdr_aodv_rrep_ack*)hdr_aodv::access(p))

/*
 * General AODV Header - shared by all formats
 */
struct hdr_aodv
{
    u_int8_t ah_type;
    // Header access methods
    static int offset_;          // required by PacketHeaderManager
    inline static int& offset() { return offset_; }
    inline static hdr_aodv* access(const Packet* p)
    {
        return (hdr_aodv*) p->access(offset_);
    }
};

struct hdr_aodv_request
{
    u_int8_t    rq_type;            // Packet Type
    u_int8_t    reserved[2];
    u_int8_t    rq_hop_count;       // Hop Count
    u_int32_t   rq_bcast_id;        // Broadcast ID

    nsaddr_t    rq_dst;             // Destination IP Address
    u_int32_t   rq_dst_seqno;       // Destination Sequence Number
    nsaddr_t    rq_src;             // Source IP Address
    u_int32_t   rq_src_seqno;       // Source Sequence Number

    double      rq_timestamp;       // when REQUEST sent;
    // used to compute route discovery latency

    #ifdef AOMDV
    nsaddr_t    rq_first_hop;       // First Hop taken by the RREQ
    #endif // AOMDV

    // This define turns on gratuitous replies-
    // see aodv.cc for implementation contributed by
    // Anant Utgikar, 09/16/02.
    //#define RREQ_GRAT_RREP 0x80

    inline int size()
    {
        int sz = 0;
        sz = 7*sizeof(u_int32_t);

        #ifdef AOMDV
        sz += sizeof(nsaddr_t);     // rq_first_hop
        #endif // AOMDV
        assert (sz >= 0);
        return sz;
    }
};
```

```
// [scheduling] ——
#ifdef SA_CHANGE
struct hdr_aodv_ping
{
    u_int8_t    type;               // packet type
    nsaddr_t    src;                // source IP address
    nsaddr_t    dst;                // destination IP address
    nsaddr_t    first_hop;          // first hop of the route
    nsaddr_t    last_hop;           // last hop of the route, used as id
    nsaddr_t    next_hop;           // next hop
    u_int8_t    hop_count;          // hop count
    int         ping_reply_flag;    // is this a reply?
    u_int8_t    nr_neigh;           // total number of neigh along the route

    inline int size()
    {
        int sz = 0;
        sz = 5*sizeof(nsaddr_t);
        sz += 3*sizeof(u_int8_t);
        sz += sizeof(int);
        assert (sz >= 0);
        return sz;
    }

};

struct hdr_aodv_info
{
    u_int8_t    type;               // Packet Type
    nsaddr_t    src;                // Source IP Address
    nsaddr_t    dst;                // Destination IP Address
    nsaddr_t    next_hop;
    nsaddr_t    first_hop;
    nsaddr_t    last_hop;
    u_int8_t    hop_count;          // Hop Count
    u_int8_t    path_count;
    double      rtt;
    int         action;

};
#endif
// [scheduling] —— end


struct hdr_aodv_reply
{
    u_int8_t    rp_type;            // Packet Type
    u_int8_t    reserved[2];
    u_int8_t    rp_hop_count;       // Hop Count
    nsaddr_t    rp_dst;             // Destination IP Address
    u_int32_t   rp_dst_seqno;       // Destination Sequence Number
    nsaddr_t    rp_src;             // Source IP Address
    double      rp_lifetime;        // Lifetime

    double      rp_timestamp;       // when corresponding REQ sent;
    // used to compute route discovery latency

    #ifdef AOMDV
    u_int32_t   rp_bcast_id;        // Broadcast ID of the corresponding RREQ
    nsaddr_t    rp_first_hop;
    #endif // AOMDV

    inline int size()
    {
        int sz = 0;
        sz = 6*sizeof(u_int32_t);
```

```
        #ifdef AOMDV
        if (rp_type == AODVTYPE_RREP)
        {
            sz += sizeof(u_int32_t);// rp_bcast_id
            sz += sizeof(nsaddr_t); // rp_first_hop
        }
        #endif // AOMDV

        assert (sz >= 0);
        return sz;
    }

};


struct hdr_aodv_error
{
    u_int8_t    re_type;            // Type
    u_int8_t    reserved[2];        // Reserved
    u_int8_t    DestCount;          // DestCount
    // List of Unreachable destination IP addresses and sequence numbers
    nsaddr_t    unreachable_dst[AODV_MAX_ERRORS];
    u_int32_t   unreachable_dst_seqno[AODV_MAX_ERRORS];

    inline int size()
    {
        int sz = 0;
        sz = (DestCount*2 + 1)*sizeof(u_int32_t);
        assert(sz);
        return sz;
    }

};

struct hdr_aodv_rrep_ack
{
    u_int8_t    rpack_type;
    u_int8_t    reserved;
};

// for size calculation of header-space reservation
union hdr_all_aodv
{
    hdr_aodv            ah;
    hdr_aodv_request    rreq;
    hdr_aodv_reply      rrep;
    hdr_aodv_error      rerr;
    hdr_aodv_rrep_ack   rrep_ack;
};

#endif /* __aodv_packet_h__ */
```

Listing A.5: aodv_packet.h

A.2. Tcl

A.2.1. aomdv.tcl

```
#----------------------------------------------------------------------
# AOMDV:TCL       FILE TO START ANS CONFIGURE  SIMULATION WITH NS
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Tcl Script]
#
# Date:     July 2006
#
# Input:
#       tracefile
#       mobility pattern
#       traffic pattern
#       queue type_
#       queue length
#       dimension x
#       dimension y
#       number of nodes
#       simulation time
#       simulation progress
#       scheduler type
#       aomdv max number of path
#
# Output:
#       [none]
#
#----------------------------------------------------------------------

# function call:
set tracefile_              [lindex $argv  0]; # tracefile
set mobility_pattern_       [lindex $argv  1]; # mobility pattern
set traffic_pattern_        [lindex $argv  2]; # traffic pattern
set queue_type_             [lindex $argv  3]; # [CMUPriQueue Queue/DropTail/PriQueue]
set queue_length_           [lindex $argv  4]; # Queue length max packet length in ifq
set dimension_x_            [lindex $argv  5]; # Dimension X
set dimension_y_            [lindex $argv  6]; # Dimension Y
set number_of_node_         [lindex $argv  7]; #
set sim_time_               [lindex $argv  8]; # Simulation duration
set sim_progress_           [lindex $argv  9]; # Simulation progess = pause_time?
set sched_type_in_          [lindex $argv 10]; # type of AOMDV scheduler
set aomdv_max_path_in_      [lindex $argv 11]; # maximum number of paths

puts "TCL--ARGUMENTS: visual test"
puts "tracefile:"
puts $tracefile_
puts "mobility pattern:"
puts $mobility_pattern_
puts "traffic pattern:"
puts $traffic_pattern_
puts "queue type:"
puts $queue_type_
puts "queue length:"
puts $queue_length_
puts "x:"
puts $dimension_x_
puts "y:"
puts $dimension_y_
puts "# nodes:"
puts $number_of_node_
```

```
puts "sim time:"
puts $sim_time_
puts "sim progress:"
puts $sim_progress_

#AOMDV special parameters
puts "sched type:"
puts $sched_type_in_
puts "max paths:"
puts $aomdv_max_path_in_

# tcl script to debug the ns
# init value in tcl/lib/ns-default

Phy/WirelessPhy set CPThresh_    10.0         ;# 10dB
Phy/WirelessPhy set CSThresh_    3.08319e-11  ;# 550m
Phy/WirelessPhy set RXThresh_    1.49226e-10  ;# 250m
Phy/WirelessPhy set bandwidth_   2e6          ;# outdated
Phy/WirelessPhy set Pt_          0.1          ;# 1mW
Phy/WirelessPhy set freq_        2.472e9      ;# 2.472GHz
Phy/WirelessPhy set L_           1.0          ;# system loss
Mac/802_11      set basicRate_   2e6          ;#
Mac/802_11      set dataRate_    54e6         ;# 54Mps

Agent/AODV      set sched_type $sched_type_in_
Agent/AODV      set aomdv_max_paths_ $aomdv_max_path_in_


set val(chan)        Channel/WirelessChannel
set val(prop)        Propagation/FreeSpace
set val(netif)       Phy/WirelessPhy
set val(mac)         Mac/802_11
set val(ifq)         $queue_type_            ;#Queue/DropTail/PriQueue
set val(ll)          LL
set val(ant)         Antenna/OmniAntenna
set val(x)           $dimension_x_           ;# X dimension of the topography
set val(y)           $dimension_y_           ;# Y dimension of the topography
set val(ifqlen)      $queue_length_          ;# max packet in ifq
set val(seed)        0.0
set val(adhocRouting) AODV
set val(nn)          $number_of_node_        ;# how many nodes are simulated
set val(cp)          $traffic_pattern_
set val(sc)          $mobility_pattern_
set val(stop)        $sim_time_              ;# simulation time
set val(progress)    $sim_progress_

# ======================================================================
# Main Program
# ======================================================================


# Initialize Global Variables

# set seed of random number generator
# (we generate random traffic and random mobility)

global defaultRNG;
$defaultRNG seed 1234;


# create simulator instance
set ns_     [new Simulator]

# setup topography object
set topo    [new Topography]

# create trace object for ns
```

```
set tracefd [open "| gawk -f tr_to_val.awk | \
        gawk -v M_N = $number_of_node_ -f val_to_sim.awk > $tracefile_.sim " w]

$ns_ use-newtrace
$ns_ trace-all $tracefd


# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#

#global node setting
set chan_1_ [new $val(chan)]
$ns_ node-config -adhocRouting $val(adhocRouting) \
        -llType $val(ll) \
        -macType $val(mac) \
        -ifqType $val(ifq) \
        -ifqLen $val(ifqlen) \
        -antType $val(ant) \
        -propType $val(prop) \
        -phyType $val(netif) \
        -channel $chan_1_ \
        -topoInstance $topo \
        -agentTrace ON \
        -routerTrace ON \
        -macTrace OFF


#  Create the specified number of nodes [$val(nn)] and "attach" them
#  to the channel.

for {set i 0} {$i <= $val(nn) } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0       ;# disable random motion
}


#
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam
for {set i 0} {$i <= $val(nn)} {incr i} {
    # 4 defines the node size in nam, must adjust it according to your scenario
    # The function must be called after mobility model is defined

    $ns_ initial_node_pos $node_($i) 4
}


#
```

```
# Tell nodes when the simulation ends
#
for {set i 0} {$i <= $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}


#
# define what happens at the end
#
proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}

$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"

puts "Starting Simulation..."
$ns_ run
```

Listing A.6: aomdv.tcl

A.3. Perl

A.3.1. topology_generator.pl

```
#!/usr/bin/perl
#----------------------------------------------------------------------------
# SCEN_GENERATOR.PL       FILE TO BUILD NODE NETWORK
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Perl Script]
#
# Date:     July 2006
#
# Input:
#       -nn [num_of_nodes]
#       -pause [pausetime]
#       -speed [maxspeed]
#       -simtime [simtime]
#       -x [maxx]
#       -y [maxy]
#       -type [type of scen]
#       -seed [random seed]
#       > [outdir/movement-file]
#
# Output:
#       Prints the coordinates of the nodes either on stdout or in the specified file.
#
# Types:
# 00: use setdest of NS
# 10: homogeneous hexagon
# 20: hexagon, nodes placed with jitter
# 30: hexagon with jitter, some nodes placed randomly, total 80 nodes
# 40: hexagon with hotspot
# 50: circular random
#
# combined with certain types, input variables as -x, -y and -nn may be ignored!
#----------------------------------------------------------------------------

# behave like a reasonable programming language
use strict;
use Switch;

# version of this script:
my $version                 = "01";

# temp folder
my $folder_temp             = "temp";
mkdir $folder_temp;

# CMU pattern generater: setdest
my $setdest="../ns2/ns-2.26/indep-utils/cmu-scen-gen/setdest/setdest";

# seed for random number generator
my $random_seed             = 0;

# type of scenario
my $scen_type               =();

# simulation duration
my $simulation_time_start   = 0;
my $simulation_time_end     = 0;
```

```
# simulation dimension
my $simulation_x_start      = 0;
my $simulation_x_end        = 0;
my $simulation_y_start      = 0;
my $simulation_y_end        = 0;

# number of nodes
my $node_number             = 0;

# movement
my $movement_pause_time     = 0; # time btw to movements
my $movement_speed          = 0; # mean speed of the node



my $DEBUG                   = 1;

my $DISTANCE                = 175;
my $NODES_FIRST_LINE        = 5;
my $JITTER                  = 0;

###################
#      MAIN       #
###################

# get arguments
argument_parsing();

#set seed of random number generator
srand($random_seed);

# select scenario
if  ($scen_type eq "00")
{   # use setdest of NS
    generate_scen_type_00();
}
elsif   ($scen_type eq "10")
{
    $DISTANCE               = 175;
    $NODES_FIRST_LINE       = 5;
    $JITTER                 = 0;
    generate_scen_type_10();
}
elsif   ($scen_type eq "20")
{
    $DISTANCE               = 175;
    $NODES_FIRST_LINE       = 5;
    $JITTER                 = 0;
    generate_scen_type_20();
}
elsif   ($scen_type eq "30")
{
    $DISTANCE               = 175;
    $NODES_FIRST_LINE       = 5;
    $JITTER                 = 0;
    generate_scen_type_30();
}
elsif   ($scen_type eq "40")
{
    $DISTANCE               = 175;
    $NODES_FIRST_LINE       = 5;
    $JITTER                 = 0;
    generate_scen_type_40()
}
elsif   ($scen_type eq "50")
{
```

```
    $DISTANCE               = 175;
    $NODES_FIRST_LINE       = 5;
    $JITTER                 = 0;
    generate_scen_type_50()
}
elsif   ($scen_type eq "XX")
{
    $DISTANCE               = 175;
    $NODES_FIRST_LINE       = 5;
    $JITTER                 = 0;
    generate_scen_type_XX()
}
else
{


    die ("scen type $scen_type not implemented\n");
}

####################
sub generate_scen_type_00
{
# call 'setdest' of ns/ns-2.26/indep-utils/cmu-scen-gen/
# output to stdout
my $command="$setdest ".
        "-n $node_number ".
        "-p $movement_pause_time ".
        "-s $movement_speed ".
        "-t $simulation_time_end ".
        "-x $simulation_x_end ".
        "-y $simulation_y_end";



print `$command`;
}

sub generate_scen_type_10
{
# generate homogeneous
#    __
#   /  \
#   \__/
#


    my @node_pos = ();
    my $dist_nodes = $DISTANCE;
    my $nn_first_line = $NODES_FIRST_LINE;
    my $n_nodes_mesh = 0;

    # create hexagon
    ($n_nodes_mesh, @node_pos) = create_hexagon($nn_first_line, $dist_nodes);

    # output
    output_ns(@node_pos);

    if($DEBUG == 1)
    {
        output_img(@node_pos);
    }
}

sub generate_scen_type_20
{
# generate homogeneous with jitter
#        _______
```

```
#       /       \
#      /  <-.->  \
#      \         /
#       \_______/
#
#

    my @node_pos = ();
    my $dist_nodes = $DISTANCE;
    my $nn_first_line = $NODES_FIRST_LINE;
    my $n_nodes_mesh = 0;

    my $beta = $JITTER;

    # place nodes
    ($n_nodes_mesh, @node_pos) = create_hexagon($nn_first_line, $dist_nodes);

    # add jitter
    for(my $a = 0; $a < $n_nodes_mesh; $a++)
    {
        $node_pos[$a][0] = $node_pos[$a][0] + ($beta * $dist_nodes * (rand(2)-1))
                                                + $dist_nodes;
        $node_pos[$a][1] = $node_pos[$a][1] + ($beta * $dist_nodes * (rand(2)-1))
                                                + $dist_nodes;
    }

    # output
    output_ns(@node_pos);

    if($DEBUG == 1)
    {
        output_img(@node_pos);
    }
}

sub generate_scen_type_30
{
# generate homogeneous with jitter and linear ...
#        _______
#       /   .   \
#      /  <-.->  \
#      \   :   . /
#       \_______/
#

    my @node_pos = ();

    my @node_mesh        = ();
    my $dist_nodes       = $DISTANCE;
    my $nn_first_line     = $NODES_FIRST_LINE;
    my $n_nodes_mesh     = 0;
    my $beta             = $JITTER;

    my @node_pos_random = ();
    my $radius          = $DISTANCE * ($NODES_FIRST_LINE - 1);
    my $nn              = 19;

    my $x = ($dist_nodes * ($nn_first_line-1)) + $dist_nodes;
    my $y = ($dist_nodes * ($nn_first_line-1))* (3**(0.5))/2+$dist_nodes;

    # create hexagon
    ($n_nodes_mesh,@node_mesh) = create_hexagon($nn_first_line, $dist_nodes);

    # add jitter
    for(my $a = 0; $a < $n_nodes_mesh; $a++)
    {
        $node_mesh[$a][0] = $node_mesh[$a][0] + ($beta*$dist_nodes*(rand(2)-1))
```

```
                                                        + $dist_nodes;
        $node_mesh[$a][1] = $node_mesh[$a][1] + ($beta*$dist_nodes*(rand(2)-1))
                                                        + $dist_nodes;
    }

    # place additional nodes within a circle
    @node_pos_random = place_nodes_uniform_rand_circle ($x, $y, $radius, $nn);

    # add the hexagon and the randomly placed nodes
    @node_pos = (@node_mesh, @node_pos_random);

    # output
    output_ns(@node_pos);

    if($DEBUG == 1)
    {
        output_img(@node_pos);
    }
}

sub generate_scen_type_40
{
#  HOTSPOT
#         _______
#        /       \
#       /  <...>  \
#       \         /
#        \_______/
#
#


    # home-jitter
    my @node_pos         = ();
    my @node_mesh        = ();
    my $dist_nodes       = $DISTANCE;
    my $nn_first_line    = $NODES_FIRST_LINE;
    my $n_nodes_mesh     = 0;
    my $beta             = $JITTER;

    # hotspot
    my @node_pos_random  = ();
    my $radius           = $DISTANCE;
    my $shift            = $DISTANCE;
    my $nn               = 19;

    my $x = ($dist_nodes * ($nn_first_line-1)) + $shift + $dist_nodes;
    my $y = ($dist_nodes * ($nn_first_line-1))* (3**(0.5)) / 2 + $shift
                                                        + $dist_nodes;

    # create hexagon
    ($n_nodes_mesh,@node_mesh) = create_hexagon($nn_first_line, $dist_nodes);

    # add jitter
    for(my $a = 0; $a < $n_nodes_mesh ; $a++)
    {
        $node_mesh[$a][0] = $node_mesh[$a][0] + ($beta * $dist_nodes * (rand(2) - 1))
                                                                + $dist_nodes;
        $node_mesh[$a][1] = $node_mesh[$a][1] + ($beta * $dist_nodes * (rand(2) - 1))
                                                                + $dist_nodes;
    }

    # place additional nodes within a circle
    @node_pos_random = place_nodes_uniform_rand_circle ($x, $y, $radius, $nn);

    # add hexagon and random nodes
    @node_pos = (@node_mesh,@node_pos_random);
```

```
    # output
    output_ns(@node_pos);

    if($DEBUG == 1)
    {
        output_img(@node_pos);
    }
}

sub generate_scen_type_50
{
    # circle 500 m
    # 80 node

    # home-jitter
    my @node_pos    = ();
    my $radius      = ($NODES_FIRST_LINE - 1) * $DISTANCE;
    my $x           = $radius + $DISTANCE;
    my $y           = $radius + $DISTANCE;
    my $node_number = 80;

    # place nodes uniformly in a circle
    @node_pos   = place_nodes_uniform_rand_circle ($x, $y, $radius, $node_number);

    # output
    output_ns(@node_pos);

    if($DEBUG == 1)
    {
        output_img(@node_pos);
    }
}

sub generate_scen_type_XX
{
# DEBUG

    my @node_pos        = ();

    my @node_mesh       = ();
    my $dist_nodes  = 125;
    my $nn_first_line= 5;
    my $n_nodes_mesh    = 0;
    my $beta            = 0.0;

    my @node_pos_random = ();
    my $radius          = 800;
    my $nn              = 0;

    my $x = ($dist_nodes * ($nn_first_line - 1)) + $dist_nodes;
    my $y = ($dist_nodes * ($nn_first_line - 1))* (3**(0.5)) / 2 + $dist_nodes;

    # make scen homogen
    ($n_nodes_mesh,@node_mesh) = create_hexagon($nn_first_line, $dist_nodes);

    # jitter
    for(my $a = 0; $a < $n_nodes_mesh ; $a++)
    {
        $node_mesh[$a][0] = $node_mesh[$a][0] + ($beta * $dist_nodes * (rand(2)-1))
                                                            + $dist_nodes;
        $node_mesh[$a][1] = $node_mesh[$a][1] + ($beta * $dist_nodes * (rand(2)-1))
                                                            + $dist_nodes;
    }

    # make random
    @node_pos_random = place_nodes_uniform_rand_circle ($x,$y,$radius,$nn);
```

```
    @node_pos = (@node_mesh, @node_pos_random);

    # output
    output_ns(@node_pos);

    if($DEBUG == 1)
    {
        output_img(@node_pos);
    }
}

sub place_nodes_uniform_rand_square
{
# @list set_node_lin_rand(x-start,x-end,y-start,y-end,n-nodes)
# generates a list with nodes that are uniform distributed over a given area

    my $x_start = $_[0];
    my $x_end   = $_[1];
    my $y_start = $_[2];
    my $y_end   = $_[3];
    my $n_nodes = $_[4];

    my @pos     = ();
    my $coor_x  = 0;
    my $coor_y  = 0;

    for (my $a = 0; $a < $n_nodes; $a++)
    {
        # position of node $a
        $coor_x = ($x_end - $x_start) * rand(1) + $x_start;
        $coor_y = ($y_end - $y_start) * rand(1) + $y_start;

        # round
        $coor_x = round_ns($coor_x);
        $coor_y = round_ns($coor_y);

        # save coordinates in the array
        $pos[$a][0] = $coor_x;
        $pos[$a][1] = $coor_y;
    }

    if ($DEBUG == 1)
    {
        print   "# set_node_lin_dist:\n";
        print   "# x start: $x_start\n";
        print   "# x end:   $x_end\n";
        print   "# y start: $y_start\n";
        print   "# y end:   $y_end\n";
        print   "# nodes:   $n_nodes\n";
    }
    return (@pos)
}

sub place_nodes_xxx_rand_circ
{
    my $x       = $_[0];
    my $y       = $_[1];
    my $radius  = $_[2];
    my $n_nodes = $_[3];

    my $angle   = 0;
    my $rand    = 0;

    my $coor_x  = 0;
    my $coor_y  = 0;
```

```
    my @pos     = ();

    for (my $a = 0; $a < $n_nodes; $a++)
    {
        # pos
        $angle  = rand(360);
        $rand   = rand($radius);

        $coor_x = $x + $rand * sin($angle);
        $coor_y = $y + $rand * cos($angle);

        # round
        $coor_x = round_ns($coor_x);
        $coor_y = round_ns($coor_y);

        # save coordinates in the array
        $pos[$a][0] = $coor_x;
        $pos[$a][1] = $coor_y;
    }

    return (@pos)
}

sub place_nodes_uniform_rand_circle
{ # place nodes randomly and uniformly distributed within a circle
    my $x       = $_[0];
    my $y       = $_[1];
    my $radius  = $_[2];
    my $n_nodes = $_[3];

    my $coor_x  = 0;
    my $coor_y  = 0;

    my $phase_one = 0;
    my $phase_sec = 0;

    my @pos = ();

    for (my $a = 0; $a < $n_nodes; $a++)
    {
        if(rand(1) >= 0.5)
        {
            $phase_one = 1;
        }
        else
        {
            $phase_one = -1;
        }
        if(rand(1) >= 0.5)
        {
            $phase_sec = 1;
        }
        else
        {
            $phase_sec = -1;
        }

        $coor_x = rand($radius);
        $coor_y = rand(($radius**2 - $coor_x**2)**0.5 );

        $coor_x = $x + $phase_one * $coor_x;
        $coor_y = $y + $phase_sec * $coor_y;

        # round
        $coor_x = round_ns($coor_x);
        $coor_y = round_ns($coor_y);
```

```
        # save coordinates in the array
        $pos[$a][0] = $coor_x;
        $pos[$a][1] = $coor_y;
    }
    return (@pos)
}

sub place_nodes_triangle
{
    my $line    = $_[0];
    my @pos     = ();

    # save coordinates in the array
    $pos[0][0] = 0;
    $pos[0][1] = 0;

    $pos[1][0] = $line;
    $pos[1][1] = 0;

    $pos[2][0] = $line / 2;
    $pos[2][1] = $line * (3**(0.5)) / 2;

    return (@pos)
}

sub create_hexagon
{ # ($node_index @pos) create_hexagon(nn_first_line,dis_bw_node)
# generates a list with nodes that are part of a mesh
# creates a hexagon with $dist distance between the nodes

    my $nn_first_line   = $_[0];    # number of nodes on the first line
    my $dist            = $_[1];    # distance btw two nodes

    my $y_delta_offset  = $dist * (3**(0.5)) / 2; # distance btw two lines of nodes
    # dist/2 * sqrt(3), height of the equilateral triangle

    my $x_offset        = ($nn_first_line - 1) * $dist / 2;
    my $x_delta_offset  = $dist / 2;

    my $node_index      = 0;
    my @pos = ();

    my $coor_x          = 0;
    my $coor_y          = 0;

    # upper part of hexagon
    my $a = 0;
    my $b = 0;

    for ($a = 0; $a < $nn_first_line; $a++) # lines
    {
        for($b = 0; $b < $nn_first_line + $a; $b++)
        {
            # position of node $node_index
            $coor_x = ($x_offset - $a * $x_delta_offset) + $b * $dist;
            $coor_y = $y_delta_offset * $a;

            # round
            $coor_x = round_ns($coor_x);
            $coor_y = round_ns($coor_y);

            # save coordinates in the array
            $pos[$node_index][0] = $coor_x;
            $pos[$node_index][1] = $coor_y;

            # next node
            $node_index++;
```

```
        }
    }

    # lower part of hexagon

    for ($a = 1 ; $a < $nn_first_line; $a++) #lines
    {
        for($b = 0; $b < (2 * $nn_first_line - 1) - $a; $b++)
        {
            # position of node $node_index
            $coor_x = ($a * $x_delta_offset) + $b * $dist;
            $coor_y = ($y_delta_offset * ($nn_first_line - 1)) + $y_delta_offset * $a;

            # round
            $coor_x = round_ns($coor_x);
            $coor_y = round_ns($coor_y);

            # save coordinates in the array
            $pos[$node_index][0] = $coor_x;
            $pos[$node_index][1] = $coor_y;

            # next node
            $node_index++;
        }
    }
    return ($node_index,@pos)
}

sub output_screen
{ # output_screen(\@node_pos)

    my @pos = @_;

    print "# node placement: \n";
    for (my $a = 0; $a < scalar(@pos); $a++)
    {
        print "# node:\t$a\t    x:\t$pos[$a][0]\t   y:\t$pos[$a][1]\n";
    }

    return
}

sub output_ns
{ # output_ns(@node_pos)

    my @pos = @_;

    print "# scen_generator - Version: $version\n";
    print "# file generated at: ",localtime(time),"\n";
    print "#\t x-coor\t  y-coor\t   z-coor \n";

    for (my $a = 0; $a < scalar(@pos); $a++)
    {
        print ("\$node_($a) set X_ $pos[$a][0];\n");
        print ("\$node_($a) set Y_ $pos[$a][1];\n");
        print ("\$node_($a) set Z_ ",round_ns(0),";\n");
    }
    return
}

sub output_img
{ # out_imag(\@node_pos)

    my @pos = @_;

    #write data file
```

```
    #print "generate gnu data file ... \n";
    open(GNUPLOT_DATA, ">$folder_temp/gnuplot_data.tmp") ||
    die "can't write gnuplot_data.tmp file";

    print GNUPLOT_DATA "#\t x-coor\t\t  y-coor\t\t   Z=1 \n";

    for(my $a = 0; $a < scalar(@pos); $a++)
    {
        print GNUPLOT_DATA "$pos[$a][0]\t\t $pos[$a][1]\t\t     1\n";
    }

    close(GNUPLOT_DATA);

    #write header file
    #print "generate gnu  plot...\n ";
    open(GNUPLOT_HEADER, ">$folder_temp/gnuplot_header.tmp") ||
    die "can't write gnuplot_header.tmp file";

    print GNUPLOT_HEADER    "
    set xrange [0:1700]
    set yrange [0:1700]
    set view map
    set size square
    set pointsize 2
    set title \"scenario generator\"
    #set noxtic
    #set noytic
    set nokey
    set output \"$folder_temp/nodes.eps\"
    set term postscript eps enhanced color
    splot \"$folder_temp/gnuplot_data.tmp\" using 1:2:3 with points 7 7" ;

    close(GNUPLOT_HEADER);

    # make graphic
    print `gnuplot $folder_temp/gnuplot_header.tmp`;

    # view plot
    print `gv $folder_temp/nodes.eps`;
}

sub round_ns
{ #round a number to 12 digits after zero

    my $in  = $_[0];
    my $out = sprintf ("%.12f", $in);

    return ($out)
}

sub argument_parsing
{ # check number of arguments and validate the input
    if(scalar(@ARGV)!= 16)
    {
        die     "use: -nn [num_of_nodes] ".
                "-pause [pausetime] ".
                "-speed [maxspeed] ".
                "-simtime [simtime] ".
                "-x [maxx] ".
                "-y [maxy] ".
                "-type [type of scen] ".
                "-seed [random seed]> ".
                "[outdir/movement-file]\n";
    }

    for(my $a = 0; $a < scalar(@ARGV); $a = $a + 2)
    {
```

```
if($ARGV[$a] eq "-nn")
{
    if($ARGV[$a+1] > 0)
    {
        $node_number = $ARGV[$a+1];
    }
    else
    {
        die "number of node $ARGV[$a+1] ?!?\n";
    }
}
elsif($ARGV[$a] eq "-seed")
{
    $random_seed = $ARGV[$a+1];
}
elsif($ARGV[$a] eq "-pause")
{
    if($ARGV[$a+1] >= 0)
    {
        $movement_pause_time = $ARGV[$a+1];
    }
    else
    {   die "pause time $ARGV[$a+1] ?!?\n";
    }
}
elsif($ARGV[$a] eq "-speed")
{
    if($ARGV[$a+1] >= 0)
    {
        $movement_speed = $ARGV[$a+1];
    }
    else
    {
        die "speed $ARGV[$a+1] ?!?\n";
    }
}
elsif($ARGV[$a] eq "-simtime")
{
    if($ARGV[$a+1] > 0)
    {
        $simulation_time_end = $ARGV[$a+1];
    }
    else
    {
        die "time $ARGV[$a+1] ?!?\n";
    }
}
elsif($ARGV[$a] eq "-x")
{
    if($ARGV[$a+1] > 0)
    {
        $simulation_x_end = $ARGV[$a+1];
    }
    else
    {
        die "x $ARGV[$a+1] ?!?\n";
    }
}
elsif($ARGV[$a] eq "-y")
{
    if($ARGV[$a+1] > 0)
    {
        $simulation_y_end = $ARGV[$a+1];
    }
    else
    {
        die "y $ARGV[$a+1] ?!?\n";
```

```
            }
        }
        elsif($ARGV[$a] eq "-type")
        {
            $scen_type = $ARGV[$a+1];
        }
        elsif($ARGV[$a] eq "-seed")
        {
            if($ARGV[$a+1] >= 0)
            {
                $random_seed = $ARGV[$a+1];
            }
            else
            {
                die "seed $ARGV[$a+1] ?!?\n";
            }
        }


        else
        {
            print " unknown argument $ARGV[$a]";
        }
    }

    if($DEBUG == 1)
    {
        print "# DEBUG: argument_parsing() \n";
        print "#  number of nodes: $node_number\n";
        print "#  random seed: $random_seed\n";
        print "#  x end $simulation_x_end\n";
        print "#  y end $simulation_y_end\n";
        print "#  scen type: $scen_type\n";
        print "#  simulation time: $simulation_time_end\n";
        print "#  movement pause time: $movement_pause_time\n";
        print "#  movement speed: $movement_speed\n";
    }
}

sub double_check
{
    my @pos_old = @_;
    my @pos_new = ();
    my $flag = 0;

    # not yet implemented


return @pos_new
}
```

Listing A.7: topology_generator.pl

A.3.2. traffic_generator.pl

```
#!/usr/bin/perl
#-----------------------------------------------------------------------------
# TRAFFIC_GENERATOR.PL       FILE TO GENERATE DIFFERENT TRAFFIC LOADS
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Perl Script]
#
# Date:     July 2006
#
# Input:
#       -type [cbr|tcp]
#       -nn [number of nodes]
#       -seed [random seed]
#       -nc [number of connections]
#       -rate [packet/s]
#       -pktsize [packet size]
#
# Output:
#       Prints the output to the stdout, creates informations about
#       connections and their packet rate and this can be used as
#       traffic model in the tcl script for NS.
#
#-----------------------------------------------------------------------------

# Behave like a reasonable programming language
use strict;

# ============================================================================
# Default Script Options
# ============================================================================

my $version                 ='01';      # version of this script

my $node_number             = 0;        # number of nodes
my $random_seed             = 0;        # random seed

my $connection_number       = 0;        # number of connections
my $connection_packet_size  = 0;        # packet size in byte
my $connection_packet_rate  = 0;        # number of packets per second
my $connection_packet_interval  = 0;    # inverse of packet rate
                                        # time between two packet
my $connection_traffic_type = 0;        # type of traffic (tcp or cbr)

my $simulation_start_time   = 0;
#my $simulation_end_time    = 1000;     # not used yet

my $warm_up_end_time        = 180;
my $warm_up_start_time      = 0;

my $max_source_per_node     = 1;        # limited number of connections per node
my $max_sink_per_node       = 1;

my $DEBUG = 0;

# ============================================================================
#
#   MAIN
#
# ============================================================================

# get arguments of traffic_generator.pl
```

```
argument_parsing();

# print header
print   "# Traffic Generator - Version: $version\n\n";
print   "# nodes: $node_number, conn: $connection_number, ".
        "send rate: $connection_packet_rate, ".
        "seed: $random_seed\n#";

# set random seed
srand($random_seed);

my $cbr_source_index        = 0; # Unique number to identify CBR source

# check if problem has a solution:
# max. connection < (max source per node)*( number node )
if($connection_number > ($node_number*$max_source_per_node) ||
    $connection_number > ($node_number*$max_sink_per_node))
{
    print " can't solve this problem ... more connections than allowed...";
    die;
}

my $src_node                = 0;
my $dst_node                = 0;
my @n_source_per_node   =();
my @n_sink_per_node  =();

# generate for each connection a src and dst
#
for (my $ a =0; $a < $connection_number; $a++)
{
    # select random source and destination
    # #Node number between (1-$node_number)

    $dst_node = int(rand($node_number));
    $src_node = int(rand($node_number));

    #check
    #   - number of established connection of the nodes
    #   - source != destination
    #   - node_number<$node_number (check boundary)

    while( ($n_source_per_node[$src_node] >= $max_source_per_node) ||
        ($n_sink_per_node[$dst_node] >= $max_source_per_node) | $dst_node == $src_node ||
        $dst_node == $node_number ||
        $src_node == $node_number)
    {
        if($DEBUG == 1)
        {
            print "src: $src_node\n";
            print "dst: $dst_node\n";
            print "number of sources of node $src_node :",
                $n_source_per_node[$src_node],
                "(CHECK:", $n_source_per_node[$src_node] >= $max_source_per_node,")\n";
            print "number of sinks of node $dst_node :",
                $n_sink_per_node[$dst_node],
                "(CHECK:",$n_sink_per_node[$dst_node] >= $max_sink_per_node,")\n";
        }
        $dst_node = int(rand($node_number));
        $src_node = int(rand($node_number));
    }

    #add new connection
    $n_source_per_node[$src_node]++;
    $n_sink_per_node[$dst_node]++;

    #create tag for NS
```

```
    if ( $connection_traffic_type eq "cbr" )
    {
        create_cbr_connection($src_node, $dst_node,$cbr_source_index);
        $cbr_source_index++;
    }
    elsif ($connection_traffic_type eq "tcp" )
    {
        die " tcp is not yet implemented\n";
    }
    else
    {
        die " unknown \$connection_traffic_type: $connection_traffic_type";
    }
}

print "#\n#  $connection_number connections established\n#";

# ======================================================================
sub argument_parsing
{
# perl traffic_generator.pl
# [-type cbr|tcp]
# [-nn nodes]
# [-seed seed]
# [-nc number of connections]
# [-rate rate]
# [-pktsize packet size]

    # check number of arguments
    if(scalar(@ARGV)!=12)
    {
        die "use: -type [cbr|tcp] ".
                "-nn [number of nodes] ".
                "-seed [random seed] ".
                "-nc [number of connections] ".
                "-rate [packet/s] ".
                "-pktsize [packet size]\n";
    }

    # input validation
    for(my $a = 0; $a < scalar(@ARGV); $a = $a + 2)
    {
        if($ARGV[$a] eq "-type")
        {
            if($ARGV[$a + 1] eq "tcp" || $ARGV[$a + 1] eq "cbr")
            {
                $connection_traffic_type = $ARGV[$a + 1];
            }
            else
            {
                die  "type $ARGV[$a+1] is not implemented\n";
            }
        }
        elsif($ARGV[$a] eq "-nn")
        {
            if($ARGV[$a+1]>0)
            {
                $node_number = $ARGV[$a + 1];
            }
            else
            {
                die "node number $ARGV[$a+1] ?!?\n";
            }
        }
        elsif($ARGV[$a] eq "-seed")
        {
            $random_seed = $ARGV[$a + 1];
```

```
        }
        elsif($ARGV[$a] eq "-nc")
        {
            if($ARGV[$a+1]>0)
            {
                $connection_number = $ARGV[$a + 1];
            }
            else
            {
                die "number of connections $ARGV[$a+1]?!?\n";
            }
        }
        elsif($ARGV[$a] eq "-rate")
        {
            if($ARGV[$a + 1] > 0)
            {
                $connection_packet_rate = $ARGV[$a + 1];
                $connection_packet_interval = 1 / $connection_packet_rate;
            }
            else
            {
                die "packet rate $ARGV[$a+1]?!?\n";
            }
        }
        elsif($ARGV[$a] eq "-pktsize")
        {
            if($ARGV[$a + 1] > 0)
            {
                $connection_packet_size = $ARGV[$a + 1];
            }
            else
            {
                die "packet size $ARGV[$a+1]?!?\n";
            }
        }

        else
        {
            print " unknown argument $ARGV[$a]";
        }

    }

    if($DEBUG == 1)
    {
        print "#  DEBUG: argument_parsing() \n";
        print "#  number of nodes: $node_number\n";
        print "#  random seed: $random_seed\n";

        print "#  traffic type: $connection_traffic_type\n";
        print "#  number of connections: $connection_number\n";
        print "#  packet interval: $connection_packet_interval \n\n";
    }
}

sub create_cbr_connection
{
# write 'NS-code' to the Stout

    my $src_node = shift;
    my $dst_node = shift;
    my $cbr_source_number = shift; # unique number to identify CBR source

    my $traffic_start_time = (rand(1) * ($warm_up_end_time - $warm_up_start_time))
                                        + $simulation_start_time;
    #my $traffic_end_time = $simulation_end_time;
```

```
    print "\n#\n# $src_node connecting to $dst_node at time $traffic_start_time\n#\n";

    # source
    print "set udp_($cbr_source_number) \[new Agent/UDP\] \n";

    print "set cbr_($cbr_source_number) \[new Application/Traffic/CBR\]\n";
    print "\$cbr_($cbr_source_number) set packetSize_ $connection_packet_size\n";
    print "\$cbr_($cbr_source_number) set interval_ $connection_packet_interval\n";
    print "\$cbr_($cbr_source_number) set random_ 1\n";
    print "\$cbr_($cbr_source_number) set maxpkts_ 10000\n";

    print "\$cbr_($cbr_source_number) attach-agent \$udp_($cbr_source_number)\n";
    print "\$ns_ attach-agent \$node_($src_node) \$udp_($cbr_source_number)\n";

    # sink
    print "set null_($cbr_source_number) \[new Agent/Null\]\n";
    print "\$ns_ attach-agent \$node_($dst_node) \$null_($cbr_source_number)\n";

    # connect source and sink
    print "\$ns_ connect \$udp_($cbr_source_number) \$null_($cbr_source_number)\n";

    # set event
    print "\$ns_ at $traffic_start_time \"\$cbr_($cbr_source_number) start\"\n";
}

# ============================================================================
```

Listing A.8: traffic_generator.pl

A.3.3. create_simulation.pl

```
#!/usr/bin/perl
#-----------------------------------------------------------------------------------------
# create_simulation.pl:
#       FILE TO GENERATE PACKAGE FOR SIMULATION SETUP
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Perl Script]
#
# Date:     July 2006
#
# Input:
#       [Parameter Settings in this file]
#
# Output:
#
#       ["datapacket.tar.gz"]
#
# This scripts creates a tar.gz packet that contains all needed files to start
# the simulation on the condor cluster of ITET
#
# Usage:
#       (0. Install remote source packet of NS on NFS of ITET)
#       (0. Adapt paths in remote.pl)

#       1. Change "System settings"
#       2. Change "Simulation Parameter"
#       3. exec: perl create_simulation.pl
#
#       (4. start condor jobs with 'condor_submit run_simulation' on ITET machine)
#       (5. check queue of condor with 'condor_q -global')
#       (6. wait until all jobs are finished or remove jobs with 'condor_rm')
#       (7. copy back result in $sim_data)
#
# Idea:
#   - create a folder under $out_dir to store all data for this $version of the simulation
#
#   - check if scenario pattern (traffic /topology) exists in $pattern_dir
#       - No -> create pattern with the help of "traffic_generator.pl" and
#           "topology_generator.pl" and save them in the $pattern_dir
#
#   - copy pattern to $out_dir
#           .
#   - write condor instruction file 'run_condor'
#       - check if data point is allready simulated?
#           --> check if .sim file exist allready in $sim_data
#               YES skip
#               NO write one condor job
#
#   - tar all data
#
#   - copy data to itet
#
#   - untar data
#
#   - start condor job with 'condor_submit run_simulation' on itet machine
#
#
# traffic generator:    see traffic_generator.pl
#
# topology generator:   see topology_generator.pl
#
```

```
# condor:                 see: www.computing.ee.ethz.ch -> programming -> condor
#
#-------------------------------------------------------------------------------

# Behave like a reasonable programming language
use strict;

#-------------------------------------------------------------------------------
# System settings
#-------------------------------------------------------------------------------

#Edit this variables to run the script on your system


# Simulation's Description
#-------------------------

# release number
my $version          = "D002";

# test number XX
my $testseries       = "002";

# description of this simulation
my $description      = "Hexagon, with hello, with info, no ping";

# Folder of the network simulator
#--------------------------------
# path to NS
my $ns_dir           = "../ns2/ns-2.26";

# Folder of simulation results
#-----------------------------
# (used to check if the point is allready simulated)
my $sim_data         = "../sim_data/$version";
mkdir $sim_data;

# Folder of simulation pattern
#-----------------------------
my $pattern_dir      = "../pattern";
mkdir $pattern_dir;

# Output
#-------
# folder for OUTPUT
my $out_dir          = "../transit";
mkdir $out_dir;

# folder for packet
my $packet_dir       = "$out_dir/$version-$testseries";
mkdir $packet_dir;

# Scripts
#--------
# this file
my $create_file      = "create_simulation";

# traffic generator
my $traffic_gen      = "traffic_generator.pl";

# topology generator
my $topology_gen     = "topology_generator.pl";

# executable of condor
my $rem_exe          = "remote.pl";

# called by remote.pl to start NS
```

```
my $rem_start_ns     = "aomdv.tcl";

# called by remote.pl to process output data
my $rem_gawk_tr_val = "tr_to_val.awk";
my $rem_gawk_sim     = "val_to_sim.awk";


# User
#-------------------------
my $user_name        = "schadomi";               # itet username


# Remote machine
#-------------------------
my $remote_host      = "tardis-do2.ee.ethz.ch";
my $remote_host_dir = "/home/schadomi/extra";   # NFS space to save the simulation


#-------------------------------------------------------------------------------------------
# Simulation Settings
#-------------------------------------------------------------------------------------------

# General
my $n_of_run         = 5;            ## number of simulations

# Variables
my @queue_type       = ('Queue/DropTail/PriQueue');  ## interface queue type
my @queue_length     = (50);         ## length of interface queue (in packets)
my @dim_x            = (2000);       ## x (network dim) --> 2000 for scene != 00
my @dim_y            = (2000);       ## y (network dim) --> 2000 for scene != 00
my @n_node           = (80);         ## number of nodes ---> 80 for scene != 00

my @sim_time         = (1000);       ## simulation duration

my @sim_progress     = (100);        ## simulation progress, not used

# Moving pattern
my @pause_time       = (0);          ## pause time (pause between moving)
my @max_speed        = ('0.0');      ## max speed [m/s]
my @scen_type        = ('40');       ## see topology_generator.pl for help

# Traffic pattern
my @source_type      = ('cbr');      ## type of traffic source ("cbr" or "tcp")
my @num_connect      = (50);         ## number of connections
my @packet_size      = (256);        ## packet size [Bytes]

my @packet_rate      = ('1.0','1.5','2.0');  ## packet rate [packet/second]

# AOMDV parameter
my @sched_type       = ('0','1','2','3');    ## type of path scheduler

                                     ## 0 = Select only one [aomdv] (shortest path)
                                     ## 1 = Round Robin [RR]
                                     ## 2 = Weighted RR [WRR] (1/hop)
                                     ## 3 = Selective WRR [SWRR]
                                     ## 4 = WRR RTT (needs ping)
                                     ## 5 = Neighbour (needs ping)
my @max_path         = (1);          ## max of paths between source and destination

#-------------------------------------------------------------------------------------------

#-------------------------
#   global variables
#-------------------------

# Pattern generation
my $seed     = 0;    # random seed for traffic generator
```

```
my @zeilen;          # array to manipulate seed
my $file_name;       # temp variable to generate path names

## loop variables:
my $n_of_run_idx         = 0;
my $queue_type_idx       = 0;
my $queue_length_idx     = 0;
my $dim_x_idx            = 0;
my $dim_y_idx            = 0;
my $n_node_idx           = 0;
my $sim_time_idx         = 0;
my $sim_progress_idx     = 0;
my $pause_time_idx       = 0;
my $max_speed_idx        = 0;
my $scen_type_idx        = 0;
my $source_type_idx      = 0;
my $num_connect_idx      = 0;
my $packet_size_idx      = 0;
my $packet_rate_idx      = 0;
my $sched_type_idx       = 0;
my $max_path_idx         = 0;

#-------------------------------------------------------------------------------

print "\n CREATE START SCRIPT \n";

#-------------------------------------------------------------------------------

print "create folder structure\n";

print `rm -r  $packet_dir`;                          # remove old data
mkdir $packet_dir;

my $packet_sim       = "$packet_dir/sim_data";       # simulation out
mkdir $packet_sim;

my $log_dir          = "$packet_dir/condor_log";     # folder to store condor logs
mkdir $log_dir;

my $log_error        = "$log_dir/error";             # condor errors
mkdir $log_error;

my $log_out          = "$log_dir/out";               # condor stdout
mkdir $log_out;

my $packet_pattern   = "$packet_dir/pattern";        # folder of transit packet
mkdir $packet_pattern;

print "copy files ...\n";

# copy executable of NS
`cp $ns_dir/ns          $packet_dir`;

# copy remote scripts
`cp $rem_exe            $packet_dir`;
`cp $rem_start_ns       $packet_dir`;
`cp $rem_gawk_tr_val    $packet_dir`;
`cp $rem_gawk_sim       $packet_dir`;

# copy myself
`cp $create_file        $packet_dir`;

print "write description ...\n";
`echo $description > $packet_dir/description.txt`;

#---------------------- Generate needed Pattern ----------------------------------
```

```
print "\n ############## START:  Generate Pattern ################## \n";

for($n_of_run_idx = 0; $n_of_run_idx < $n_of_run; $n_of_run_idx++) #runs
{
 print "-----------------------------------------------------------\n";
 print " run number: $n_of_run_idx\n";
 print "-----------------------------------------------------------\n";

 for($dim_x_idx = 0; $dim_x_idx < @dim_x; $dim_x_idx++)
 { print "\n* dimension x:          \t\t$dim_x[$dim_x_idx]\n";

  for($dim_y_idx = 0; $dim_y_idx < @dim_y; $dim_y_idx++)
  { print "\n* dimension y:          \t\t$dim_y[$dim_y_idx]\n";

   for($n_node_idx = 0; $n_node_idx < @n_node; $n_node_idx++)
   { print "\n* number of nodes:      \t\t$n_node[$n_node_idx]\n";

    for($sim_time_idx = 0; $sim_time_idx < @sim_time; $sim_time_idx++)
    { print "\n* simulation duration:\t\t$sim_time[$sim_time_idx]\n";


    # make folders
    #-------------------------
    print "\n\t- make folders ... \n";

    # local
    # generate pattern folder if it does not exist
    mkdir "$pattern_dir";
    mkdir "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]-$sim_time[$sim_time_idx]";
    mkdir "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]-$sim_time[$sim_time_idx]".
            "/mobility-$n_of_run_idx";
    mkdir "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]-$sim_time[$sim_time_idx]".
            "/sources-$n_of_run_idx";

    # packet
    # generate pattern folder if does not exist

    mkdir "$packet_pattern/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
        "-$sim_time[$sim_time_idx]";
    mkdir "$packet_pattern/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
        "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx";
    mkdir "$packet_pattern/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
        "-$sim_time[$sim_time_idx]/sources-$n_of_run_idx";


    # set random seed
    #-------------------------
    print "\t- set random seed ...\n";
    $seed = $n_of_run_idx + 1;

    # overflow check
    if($seed > 32767)
    {
        die ("running out of seeds (bigger then int16 )...\n");
    }
    elsif($seed == 0)
    {
        die ("BUG: traffic generator does not work with seed == 0\n ");
    }

    #Check if seed.txt file exists
    if (1 == -e "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
            "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx/seed.txt")
    {
        # read seed out of file
        print "\t\t\t-- read seed out of file: ";
        open(FILE,"<$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
```

```
        "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx/seed.txt") ||
    die "can't load seed file";

    @zeilen = <FILE>;
    close(FILE);

    $seed = $zeilen[0];
    print "$seed \n";
}
else
{
    #set seed and write it into the
    #   "mobility-RUN/seed.txt" and "sources-RUN/seed.txt"

    print "\t\t\t-- generate seed and store it in file: ";
    open(FILE,">$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
        "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx/seed.txt") ||
    die "can't write seed file";

    print "$seed\n";

    print FILE $seed;
    close(FILE);

    #copy file in source folder
    my $command ="cp $pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
                        "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx/seed.txt ".
                    "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
                        "-$sim_time[$sim_time_idx]/sources-$n_of_run_idx/";
    print `$command`;

}


# generate pattern
#-------------------------
print "\t- generate pattern ...\n\n";

# topology
#-------------------------
print"\t\tTOP-PAT\n";

for($pause_time_idx = 0; $pause_time_idx < @pause_time; $pause_time_idx++)
    { print "\t\tpause time: $pause_time[$pause_time_idx]\n";

 for($max_speed_idx = 0; $max_speed_idx < @max_speed; $max_speed_idx++)
     { print "\t\t\tmaximum speed: $max_speed[$max_speed_idx]\n";

    for($scen_type_idx = 0; $scen_type_idx < @scen_type; $scen_type_idx++)
        { print "\t\t\t\tscen type: $scen_type[$scen_type_idx]\n";

    #file name
    $file_name = "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
        "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx/scen-$n_node[$n_node_idx]".
        "-$pause_time[$pause_time_idx]-$max_speed[$max_speed_idx]".
        "-type-@scen_type[$scen_type_idx]";

    #Check if file exist
    if (1== -e $file_name)
    {
        print "\t\t\t\t top: file $file_name exist \n ";
    }
    else
    {
        print "\t\t\t\t top: generate $file_name \n";

        my $command="perl $topology_gen ".
```

```
                    "-nn $n_node[$n_node_idx] ".
                    "-pause $pause_time[$pause_time_idx] ".
                    "-speed $max_speed[$max_speed_idx] ".
                    "-simtime $sim_time[$sim_time_idx] ".
                    "-x $dim_x[$dim_x_idx] ".
                    "-y $dim_y[$dim_y_idx] ".
                    "-type $scen_type[$scen_type_idx] ".
                    "-seed  $seed > $file_name";
        print `$command `;
    }

    # copy file into rem dir
    my $command ="cp -f $file_name ".
                "$packet_pattern/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
                "-$sim_time[$sim_time_idx]/mobility-$n_of_run_idx";
    print `$command`;
   }#scen type
  }#max speed
 }#pause
 #end movement

# traffic
#-------------------------
print"\t\tTRA-PAT\n";

 for($source_type_idx = 0; $source_type_idx < @source_type; $source_type_idx++)
      { print "\t\tsource type: $source_type[$source_type_idx]\n";

   for($num_connect_idx = 0; $num_connect_idx < @num_connect; $num_connect_idx++)
       { print "\t\t\tnumber of connections: $num_connect[$num_connect_idx]\n";

    for($packet_size_idx = 0; $packet_size_idx < @packet_size; $packet_size_idx++)
        { print "\t\t\t\tpacket_size: $packet_size[$packet_size_idx]\n";

      for($packet_rate_idx = 0; $packet_rate_idx < @packet_rate; $packet_rate_idx++)
          { print "\t\t\t\t\tpacket_rate: $packet_rate[$packet_rate_idx]\n";

        # out file name
        $file_name = "$pattern_dir/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
                "-$sim_time[$sim_time_idx]/sources-$n_of_run_idx/".
                "$source_type[$source_type_idx]-$n_node[$n_node_idx]".
                "-$num_connect[$num_connect_idx]-$packet_rate[$packet_rate_idx]-".
                "$packet_size[$packet_size_idx]";
        # check if file exist
        if (1== -e $file_name)
        {
            print "\t\t\t\t traffic: file $file_name already exist \n ";
        }
        else
        {
            print "\t\t\t\t traffic: generate $file_name  \n";
            # System Call
            #
            my $command =   "perl $traffic_gen ".
                            "-type $source_type[$source_type_idx] ".
                            "-nn $n_node[$n_node_idx] ".
                            "-seed $seed ".
                            "-nc $num_connect[$num_connect_idx] ".
                            "-pktsize $packet_size[$packet_size_idx] ".
                            "-rate $packet_rate[$packet_rate_idx]".
                            " > $file_name";
            print `$command`;
        }

        # copy file into rem dir
        my $command = "cp -f $file_name ".
            "$packet_pattern/$dim_x[$dim_x_idx]-$dim_y[$dim_y_idx]".
```

```
                "-$sim_time[$sim_time_idx]/sources-$n_of_run_idx";
            print `$command`;
        }#packet rate
       }#packet size
      }#max connect
     }#source type
     #end traffic
    }#simulation duration
   }# n of nodes
  }#dim y
 }#dim x
}#n of run


print "\n\n ######################## GENERATE CONDOR FILE ############ \n \n";

# open file to write
my $condor_filename = "$packet_dir/run_simulation.condor";
open(FILE, ">$condor_filename");  # Open for out

# write XXX.condor
print FILE "#CONDOR SUBMIT FILE\n";
print FILE
"
#-----------------------------------------------------------------------------
#
# Project:  Semester Project SS 05/06:
#           'Evaluation of scheduling methods over multipath
#               routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#       [condor file - auto generated]
#
#
# built by \"create_simulation.pl\" perl script!
#-----------------------------------------------------------------------------

#START CONDOR

universe    = vanilla
getenv      = True          # 'Network Simulator' needs local environment
Rank        = Kflops

# Problem: code is compiled for linux --> use only \"LINUX\"
Requirements = ( (Arch == \"INTEL\" || Arch == \"PPC\" || Arch == \"x86_64\" || ".
                "Arch == \"ALPHA\" ||  Arch == \"SUN4u\" ) ".
                "&& (OpSys == \"LINUX\" ) ".
                "&& Memory >=32 )"."

#Only send a Email when
Notification = Error

# Let jobs run under TIK rights
# (... even if I'm a student with only ITET rights ...) ;-)
+USER_GROUP = \"tik\"
+JOB_GROUP = \"tik\"

# executable File
executable  = $rem_exe

\n\n\n";


my $process  = 0;  # counter of processes

print "\n\n";
```

```
for($n_of_run_idx = 0; $n_of_run_idx < $n_of_run  ; $n_of_run_idx++)
{print "run number: $n_of_run_idx\n";

 for($queue_type_idx = 0; $queue_type_idx < @queue_type; $queue_type_idx++)
 {print " queue type: $queue_type[$queue_type_idx]\n";

  for($queue_length_idx = 0; $queue_length_idx < @queue_length; $queue_length_idx++)
  {print "  queue length: $queue_length[$queue_length_idx]\n";

   for($dim_x_idx = 0; $dim_x_idx < @dim_x; $dim_x_idx++)
   {print "   dimension x: $dim_x[$dim_x_idx]\n";

    for($dim_y_idx = 0; $dim_y_idx < @dim_y; $dim_y_idx++)
    {print "    dimension y: $dim_y[$dim_y_idx]\n";

     for($n_node_idx = 0; $n_node_idx < @n_node; $n_node_idx++)
     {print "     number of nodes: $n_node[$n_node_idx]\n";

      for($sim_time_idx = 0; $sim_time_idx < @sim_time; $sim_time_idx++)
      {print "      simulation duration: $sim_time[$sim_time_idx]\n";

       for($sim_progress_idx = 0; $sim_progress_idx < @sim_progress; $sim_progress_idx++)
       {print "       simulation progress time: $sim_progress[$sim_progress_idx]\n";

        for($pause_time_idx = 0; $pause_time_idx < @pause_time; $pause_time_idx++)
        {print "        pause time: $pause_time[$pause_time_idx]\n";

         for($max_speed_idx = 0; $max_speed_idx < @max_speed; $max_speed_idx++)
         {print "         maximum speed: $max_speed[$max_speed_idx]\n";

          for($source_type_idx = 0; $source_type_idx < @source_type; $source_type_idx++)
          {print "          source type: $source_type[$source_type_idx]\n";

           for($num_connect_idx = 0; $num_connect_idx < @num_connect; $num_connect_idx++)
           {print "           num_connections: $num_connect[$num_connect_idx]\n";

            for($packet_size_idx = 0; $packet_size_idx < @packet_size; $packet_size_idx++)
            {print "            packet_size: $packet_size[$packet_size_idx]\n";

for($packet_rate_idx = 0; $packet_rate_idx < @packet_rate; $packet_rate_idx++)
{print "             packet_rate: $packet_rate[$packet_rate_idx]\n";

 for($sched_type_idx = 0; $sched_type_idx < @sched_type; $sched_type_idx++)
 {print "              sched_type: $sched_type[$sched_type_idx]\n";

   for($max_path_idx = 0; $max_path_idx < @max_path; $max_path_idx++)
   {print "               max_path: $max_path[$max_path_idx]\n";

    for($scen_type_idx = 0; $scen_type_idx < @scen_type; $scen_type_idx++)
    {print "               scen_type(pat): $scen_type[$scen_type_idx]\n";

    print ("condor filecheck: ", -s "$sim_data".
                "-ri-$n_of_run_idx".
                "-qt-$queue_type_idx".
                "-ql-$queue_length[$queue_length_idx]".
                "-dx-$dim_x[$dim_x_idx]".
                "-dy-$dim_y[$dim_y_idx]".
                "-nn-$n_node[$n_node_idx]".
                "-sd-$sim_time[$sim_time_idx]".
                "-sp-$sim_progress[$sim_progress_idx]".
                "-pt-$pause_time[$pause_time_idx]".
                "-ms-$max_speed[$max_speed_idx]".
                "-scp-$scen_type[$scen_type_idx]".
                "-st-$source_type[$source_type_idx]".
                "-mc-$num_connect[$num_connect_idx]".
                "-ps-$packet_size[$packet_size_idx]".
```

```
            "–pr–$packet_rate[$packet_rate_idx]".
            "–sc–$sched_type[$sched_type_idx]".
            "–pn–$max_path[$max_path_idx].sim","\n");

#Check if datapoint (*.sim) allready exists
if((–s "$sim_data".
        "–ri–$n_of_run_idx".
        "–qt–$queue_type_idx".
        "–ql–$queue_length[$queue_length_idx]".
        "–dx–$dim_x[$dim_x_idx]".
        "–dy–$dim_y[$dim_y_idx]".
        "–nn–$n_node[$n_node_idx]".
        "–sd–$sim_time[$sim_time_idx]".
        "–sp–$sim_progress[$sim_progress_idx]".
        "–pt–$pause_time[$pause_time_idx]".
        "–ms–$max_speed[$max_speed_idx]".
        "–scp–$scen_type[$scen_type_idx]".
        "–st–$source_type[$source_type_idx]".
        "–mc–$num_connect[$num_connect_idx]".
        "–ps–$packet_size[$packet_size_idx]".
        "–pr–$packet_rate[$packet_rate_idx]".
        "–sc–$sched_type[$sched_type_idx]".
        "–pn–$max_path[$max_path_idx].sim")==0)
{ #*.sim doesn't exist –> simulate datapoint

    print FILE #change out to file
                "
    #JOB NUMBER:\t $process\n

    arguments = $version–".              #name of sim file
                "ri–$n_of_run_idx–".
                "qt–$queue_type_idx–".
                "ql–$queue_length[$queue_length_idx]–".
                "dx–$dim_x[$dim_x_idx]–".
                "dy–$dim_y[$dim_y_idx]–".
                "nn–$n_node[$n_node_idx]–".
                "sd–$sim_time[$sim_time_idx]–".
                "sp–$sim_progress[$sim_progress_idx]–".
                "pt–$pause_time[$pause_time_idx]–".
                "ms–$max_speed[$max_speed_idx]–".
                "scp–$scen_type[$scen_type_idx]–".
                "st–$source_type[$source_type_idx]–".
                "mc–$num_connect[$num_connect_idx]–".
                "ps–$packet_size[$packet_size_idx]–".
                "pr–$packet_rate[$packet_rate_idx]–".
                "sc–$sched_type[$sched_type_idx]–".
                "pn–$max_path[$max_path_idx] ".

    #remote exe
    "$rem_start_ns ".

    #topology
    "pattern/$dim_x[$dim_x_idx]–$dim_y[$dim_y_idx]–$sim_time[$sim_time_idx]/".
        "mobility–$n_of_run_idx/scen–$n_node[$n_node_idx]–".
        "$pause_time[$pause_time_idx]–$max_speed[$max_speed_idx]–".
        "type–@scen_type[$scen_type_idx] ".

    #traffic
    "pattern/$dim_x[$dim_x_idx]–$dim_y[$dim_y_idx]–$sim_time[$sim_time_idx]/".
        "sources–$n_of_run_idx/$source_type[$source_type_idx]–".
        "$n_node[$n_node_idx]–$num_connect[$num_connect_idx]–".
        "$packet_rate[$packet_rate_idx]–$packet_size[$packet_size_idx] ".

    # other parameters ...
    "$queue_type[$queue_type_idx] ".          # queue type
    "$queue_length[$queue_length_idx] ".      # queue length
    "$dim_x[$dim_x_idx] ".                    # dimension x
```

```
        "$dim_y[$dim_y_idx] ".                      # dimension y
        "$n_node[$n_node_idx] ".                    # number of nodes
        "$sim_time[$sim_time_idx] ".                # simulation duration
        "$sim_progress[$sim_progress_idx] ".        # simulation progress
        "$sched_type[$sched_type_idx] ".            # scheduler type
        "$max_path[$max_path_idx]".     # max number of paths

        # where to save the log files
        "\n
        log        = condor_log/condor.log
        out     = condor_log/out/$process
        error      = condor_log/error/$process
        queue\n\n";

        $process++;
    }

                }# scen_type
               }# aomdv max path
              }# scheduler type
             }# packet rate
            }# packet size
           }# max connestion
          }# source type
         }# max speed
        }# pause time
       }# simulation progress
      }# simulation duration
     }# n of nodes
    }# dim y
   }# dim x
  }# queue length
 }# queue type
}# run

close (FILE);

chdir "$out_dir";

#tar folder and upload
print "tar folder ... \n";
print `tar -czvf transit_$version-$testseries.gz.tar $version-$testseries`;
print("upload to ITET? to $remote_host_dir \n (Y/N)\n");

my $input = <STDIN>;
chomp $input;

if($input eq 'Y' || $input eq 'y')
{
    print("Warning: Sometimes scp stops!?! ... ssh bug ?\n" );
    `scp transit_$version-$testseries.gz.tar $user_name\@$remote_host:$remote_host_dir`;

    # decompress folder
    print "Decompress tar on ITET to  $remote_host_dir? \n Y/N\n";

    $input = <STDIN>;
    chomp $input;

    if($input eq 'Y' || $input eq 'y')
    {
        my $command="ssh $user_name\@$remote_host ".
            "tar xzvf $remote_host_dir/transit_$version-$testseries.gz.tar ".
            "--directory $remote_host_dir";
        print `$command`;
    }
}
```

```
print "\n\n Master, I finished my work \n\n";
```

Listing A.9: create_simulation.pl

A.3.4. remote.pl

```
#!/usr/bin/perl
#-----------------------------------------------------------------------
# REMOTE.PL       file to build condor submit file
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Perl Script]
#
# Date:     July 2006
#
# Input:
#       [none]
# Output:
#
#-----------------------------------------------------------------------

#
#
# Script is executed on the remote host
#

# Behave like a reasonable programming language
use strict;

print "\n\n";
print "################################  \n";
print "#  remote scripts started       \n";
print "#  remote.pl                    \n";
print "################################  \n";

print "\n\n--- parameter list ---\n";

my $name_of_process = $ARGV[0];
my $tcl_script      = $ARGV[1];
my $move_pattern    = $ARGV[2];
my $traffic_pattern = $ARGV[3];
my $queue_type      = $ARGV[4];
my $queue_length    = $ARGV[5];
my $dimension_x     = $ARGV[6];
my $dimension_y     = $ARGV[7];
my $number_of_node  = $ARGV[8];
my $sim_time        = $ARGV[9];
my $sim_progress    = $ARGV[10];
my $sched_type      = $ARGV[11];
my $aomdv_max_path  = $ARGV[12];

#cout parameter (visual feedback check)
print "name of process:          \t\t $name_of_process \n";
print "name of tcl script:       \t\t $tcl_script\n";
print "name of movement pattern: \t\t $move_pattern \n";
print "name of traffic pattern:  \t\t $traffic_pattern \n";
print "type of interface queue : \t\t $queue_type \n";
print "length of interface queue:\t\t $queue_length\n";
print "dimension x:              \t\t $dimension_x\n";
print "dimension y:              \t\t $dimension_y\n";
print "number of nodes:          \t\t $number_of_node\n";
print "simulation duration:      \t\t $sim_time\n";
print "simulation progress:      \t\t $sim_progress\n";     # unused ...
print "scheduling type:          \t\t $sched_type\n";
print "max number of paths       \t\t $aomdv_max_path\n";

print "start NS ...\n\n";
```

```
#-----------------------------------
#
# start NS and set working paths
#
#-----------------------------------

my $command =
    "echo \"set path\" ;".
    "export PATH=\$HOME/ns-small-2.26/bin:".
                "\$HOME/ns-small-2.26/tcl8.3.2/unix:".
                "\$HOME/ns-small-2.26/tk8.3.2/unix:".
                "\$PATH;".
    "export TCL_LIBRARY=\$HOME/ns-small-2.26/tcl8.3.2/library;".
    "export LD_LIBRARY_PATH=\"\$HOME/ns-small-2.26/lib:".
                            "\$HOME/ns-small-2.26/otcl-1.0a8\";".
    "echo \"start NS\";".
    "./ns $tcl_script sim_data/$name_of_process $move_pattern ".
        "$traffic_pattern $queue_type $queue_length $dimension_x ".
        "$dimension_y $number_of_node $sim_time $sim_progress ".
        "$sched_type $aomdv_max_path";

print(`$command`);

#-----------------------------------
#
# process output data
#
#-----------------------------------

# Due to TCL 2.0 GB file-size bug, the data are precessed on the fly
# This is implemented in the aomdv.tcl

#-----------------------------------
#
# finishing
#
#-----------------------------------

# wait until gawk has finished his job

print("check if gawk has finished his job...\n");

my $trigger = 1;
my $time_out = 6;

while($trigger)
{
    if( -z "sim_data/$name_of_process.sim") #check filesize
    {
        # file has zero size
        # just wait ...
        print "gawk is still working\n";

        if($time_out <= 0) #problems?
        {
            print"time_out ... stop simulation ...\n";
            $trigger = 0;
        }
        else
        {
            $time_out = $time_out - 1;
            sleep(10);
        }
    }
    else
    {
```

```
        # gawk process is finished
        print "gawk has finished his job";
        $trigger = 0;
        sleep(10);
    }
}
print "\nnext job please ... \n"
```

Listing A.10: remote.pl

A.3.5. analysis_one_run.pl

```
#!/usr/bin/perl
#-------------------------------------------------------------------------------
# ANALYSIS_ONE_RUNS.PL       FILE TO GENERATE ANALYSE ONE .SIM FILE
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Perl Script]
#
# Date:     July 2006
#
# Input: [-file SIM-FILE]
#
#
# Output: graphics in eps format
#
#
# Idea:
#       This scripts shows different information about this (sim) run
#           - show the number of found paths at the source over the
#                   time for one connection
#           - calculate the average number of paths for all connections
#           - shows all paths from source to destination
#
#
# Concept:
#       1. Read in the information of one .sim file
#       2. Calculate all needed variables for the statistic
#           - time discretisation based on events (time slots)
#           - calculate the active paths for each time slot
#       3. Calculate the statistics
#           - number of found path on average for each connection
#       4.  Menu to select which information should be displayed
#
# Implementation:
#
#       sub argument_parsing        fetch arguments of the command line
#
#       sub menu                    user interface
#
#       sub import_data         read data out of .sim file into hash tables
#
#       sub number_sort         helper function for sort() function call
#
#       sub calculate_stat_rt_s_t   calculate all needed information based on
#                                   the routing table space time (path)
#                                       - number of paths at time X
#                                       - number of paths in avg
#                                       - number of paths found in routing table
#                                           from source to destination at time X
#
#       sub calculate_stat_rt_src   calculate all needed information based on
#                                   the routing table source t>180
#                                       - number of paths in avg
#                                       - max number of paths over time
#                                       - number of paths found by one route request
#
#       sub sub dropzone            plots the number of dropped/sent packet in an area
#
#       sub calc_stat_shortest      calculate all statistics based on the shortest path
#
#           sub calc_stat_shortest_connection   : distribution of 'shortest path'
#                                                 in this simulation
```

```
#           sub calc_stat_180_time              : avg found path over time
#                                                     versus 'shortest path'
#           sub calc_stat_180_rp_p              : avg found path for one RREQ
#                                                     versus 'shortest path'
#           sub plot_path_shortest              : plots the statistic over shortest paths
#
#       sub connection_info         user interface for one connection
#           sub show_path                       : shows all active paths during the
#                                                     simulation as text
#           sub make_movie                      : makes an animated gif out of
#                                                      the path information
#           sub make_movie_short                : generates an animated gif out of
#                                                     a subset of the simulation
#           sub plot_path_time                  : plots the number of found paths
#                                                     for one connection over time
#           sub plot_path_space                 : plots the path in space
#
#-------------------------------------------------------------------------------

use strict;

# ------------------------------------
#
# global variables
#
# ------------------------------------

# switch DEBUG OUTPUT on/off
my $DEBUG            = 0;

# path of whirlgif (make animated gif)
my $whirlgif ="/home/alpha/sa/svn-dir/trunk/Sources/whirlgif/whirlgif";

# folder to save temporal data files
my $folder_temp          = "temp";
mkdir $folder_temp;

# .sim file with the statistic (ARG)
my $sim_file = "";

#----- DATA CONTAINER #-----
# based of .sim file

# simulation end
#-------------------------------------#
# ! ! ! Assumption simulation end = last data packet sent
#-------------------------------------#
my $simulation_end           = 0;

# number of nodes
my $node_total               = 0;

# position of nodes
my @position_x_node          = ();
my @position_y_node          = ();

# dropped packet
my @dropped_packet_data_node     = ();
my $dropped_packet_data_total    = 0;

my @dropped_dpackets_v_tot_sent      =();
my @dropped_dpackets_v_received_node     =();

# sent packet
my @sent_packet_data_node    = ();
my $sent_packet_data_total   = 0;
```

```
# received packet
my @received_packet_data_node   = ();
my $received_packet_data_total  = 0;

# forwarded packet
my @forwarded_packet_data_node  = ();

# ratio:
my $data_delivery_ratio  = 0;

# check if trace was complete
my $data_packet_first_time =0;
my $data_packet_last_time =0;


# based on rt_s_t

# database with all data connections
my %src_dest                                = ();
# $src_dest{"num con"}                      = # number of connections (1-xx)
# $src_dest{$src}{$dst}                     = $connection_index;
# $src_dest{# connection_index}{"src"}      = $src;
# $src_dest{# connection_index}{"dst"}      = $dst;

# database with all paths of the data connections
my %src_dest_path                           =();
# $src_dest_path{$src}{$dst}{"num paths"}   = #number of paths
# $src_dest_path{$src}{$dst}{$path_index}   = $path;
# $src_dest_path{$src}{$dst}{$path}         = $path_index;

# timeline discretisation
my @timeline_up                             =();
# $timeline_up[$src][$dst]                  = [sort(number_sort @timeline_up_temp)];
# --> Ref on array ... use @{Ref} to access data

my @timeline_down                           =();
# $timeline_down[$src][$dst]                = [sort(number_sort @timeline_down_temp)];
# --> Ref on array ... use @{Ref} to access data

# event database
my %event_hash                              =();

# $event_hash{$src}{$dst}{"up"}{$uptime}{"num events"} = number of events at time x

# $event_hash{$src}{$dst}{"up"}{$uptime}{ # event } = $path

# $event_hash{$src}{$dst}{"down"}{$downtime}{"num events"} = number of events at time x

# $event_hash{$src}{$dst}{"down"}{$downtime}{ # event } = $path

# based on rt_src

# what the source saw
my %path_time_src                           =();
#$path_time_src{$src}{$dst}{"shortest path"} = length of shortest paths of this src dest

#$path_time_src{$src}{$dst}{"num slots"} = Number of time time slots

#$path_time_src{$src}{$dst}{#slot}{"time"} = real time of this time slot

#$path_time_src{$src}{$dst}{#slot}{"num paths"} = number of active paths at this time slot


#------ CALCULATED DATA's #----------------------

my %path_time_s_t       =(); # Paths over the time based on routing table infos
# $path_time{$src}{$dst}{"num slots"};
```

```
my @path_time_s_t_active_paths  = (); # active paths at #slot
# path_time_s_t_active_paths[$src][$dst][#slot] = ref-> array with active paths

#routing protocol performance data container
my @rp_performace  = ();
#$rp_performance [# shortest path][# number of found paths] = $counter
#$rp_performance [# shortest path][ 0 ] = $total_counter


#------ STATISTIC CONTAINER #----------------------
my %path_shortest        =(); # ave paths found for this shortest path


#-------------------------------------------------------------------------------
#
#   main
#
#-------------------------------------------------------------------------------

print "
#-------------------------------------------------------------
\n
SEMESTER PROJECT (SS 06):
TITLE:   Evaluation of scheduling methods over multipath
         routing in wireless mobile Ad Hoc networks
AUTHOR:  Regula Goenner [rgoenner\@ee.ethz.ch]
         Dominik Schatzmann [schadomi\@ee.ethz.ch]
TUTOR:   Georgios Parissidis
\n
#-------------------------------------------------------------
\n";

print "       #-----------------------------------------#\n";
print "       # Script-BOX Version 001                  #\n";
print "       #-----------------------------------------#\n";
print "\n";

argument_parsing();

printf("import data of .sim file ...\n");
import_data();

printf("calculate statistics ...\n");

for(my $ii = 1; $ii <= $src_dest{"num con"}; $ii++)
{
    printf("\t connection %i \n",$ii);
    calculate_stat_rt_s_t($ii);
    calculate_stat_rt_src($ii);
}

calc_stat_shortest();

menu();

#-------------------------------------------------------------------------------
#
#   subfunctions
#
#-------------------------------------------------------------------------------

sub argument_parsing
{
    # read arguments into variables

    if(scalar(@ARGV)!= 2)
    {
```

```
        die( "use: -file [SIM FILE] \n");
    }
    if($ARGV[0] eq "-file")
    {
        $sim_file = $ARGV[1];
    }
    else
    {
        die("$ARGV[1] is not a valid input parameter\n")
    }
}

sub menu
{   # User Interface

    my $menu = 1;
    while($menu)
    {
        print "\n\n";
        print " ------------  \n";
        print " |  MENU  |  \n";
        print " ------------  \n";

        print "File: $sim_file\n";
        print "Delivery ratio (received/sent data packets): $data_delivery_ratio\n";
        print "\n";
        print "CHOOSE: \n";
        print "Information over one connection   (submenu)        1\n";
        print "Information over all connections  (submenu)        2\n";
        print "List all connections              (submenu)        3\n";
        print "EXIT                                               0\n";

        $menu = <STDIN>;
        chomp $menu;

        if($menu == 1)
        { # one connection
            printf("Connection number ?\n");
            my $connection_=0;
            $connection_ = <STDIN>;
            chomp $connection_;

            if($connection_> 0 && $connection_ <= $src_dest{"num con"} )
            {
                connection_info($connection_);
            }
        }
        elsif($menu == 2)
        { # all connections

            my $menu_all = 1;

            while($menu_all)
            {
                print "\n\n";
                print " ## ALL CONNECTIONS ### \n";
                print " #------------------------------------------ ";
                print "\n";
                print "Number of found paths per link (normal simulation) 1\n";
                print "Dropzone                                           2\n";
                print "RETURN                                             0\n";

                $menu_all = <STDIN>;
                chomp $menu_all;

                if($menu_all == 1)
                { #Number of found paths per link
```

```
                    plot_path_shortest();
                }
                elsif($menu_all == 2)
                { #Dropzone
                    dropzone();
                }
                elsif($menu_all == 0)
                { #RETURN

                }
                else
                {
                    print "unknown menu: $menu_all\n";
                }
            }
        }
        elsif($menu == 3)
        {  # Display all connections
            printf("\nCONNECTIONS    total: %s\n",$src_dest{"num con"});

            for(my $ii = 1; $ii <= $src_dest{"num con"}  ; $ii++)
            {
                my $src              = $src_dest{$ii}{"src"} ;
                my $dst              = $src_dest{$ii}{"dst"};
                my $shortest_path    = $path_time_src{$src}{$dst}{"shortest path"};

                printf("NR %i  \t Src: %i  \t Dest: %i  \t Shortest path: %i  \n",
                        $ii,    $src,       $dst,       $shortest_path);
            }
        }
        elsif($menu == 0)
        { # end
            $menu=0
        }
        else
        { #unknown parameter
            print "unknown menu: $menu\n";
        }

    }#while $menu
}# END sub menu

sub import_data
{ # read data out of .sim file

    #local variable
    my @lines    = ();
    my $line     = 0;
    my @elements     = ();

    # loop variable over all nodes
    my $node_index  = 0;

    # open sim file
    open(SIM_FILE, "< $sim_file") or die "Couldn't open  $sim_file \n";

    # read file into array @lines
    (@lines) = <SIM_FILE>;

    foreach $line (@lines)  # for each line
    {
        # split line in elements
        @elements = split(/ /, $line);

        # parse each line
        if ( ($elements[0] =~ m/(0|1|2|3|4|5|6|7|8|9)/)  && ( scalar(@elements) > 3 )  )
        {
```

```
    # node line

    # sent data packet
    $sent_packet_data_node[$node_index]         = $elements[3];
    $sent_packet_data_total                     +=$elements[3];

    # received data packet (AG)
    $received_packet_data_node[$node_index]     = $elements[5];
    $received_packet_data_total                 +=$elements[5];

    # forwarded data packet (RTR)
    $forwarded_packet_data_node[$node_index]    = $elements[6];

    # dropped data packet
    $dropped_packet_data_node[$node_index]      = $elements[8];
    $dropped_packet_data_total                  +=$elements[8];

    # node position
    $position_x_node[$node_index]               = $elements[1];
    $position_y_node[$node_index]               = $elements[2];

    # increment node counter
    $node_index++;
    $node_total=$node_index;
}
elsif ($elements[0] =~ /RATIO/)
{
    # delivered data
    $data_delivery_ratio = $elements[1];
}
if ($elements[0] =~ /first data with id/ && scalar(@elements) < 8)
{
    # first data packet sent
    $data_packet_first_time= $elements[3];
}
if ($elements[0] =~ /last data with id/ && scalar(@elements) < 8)
{
    # last data packet sent
    $data_packet_last_time= $elements[3];

    # ASSUMPTION
    $simulation_end = $data_packet_last_time;
}

elsif ($elements[0] =~ "#RT-H-S-T")
{
    # routing table space time header

    #print ($line);
}
elsif ($elements[0] =~ '#RT-S-T')
{
    # routing table space time data

    # interface definition
    my $src             = $elements[1];
    my $dst             = $elements[2];
    my $path_number     = $elements[3];
    my $path            = $elements[4];
    my $up              = $elements[5];
    my $down            = $elements[6];


    # temporal array to construct time line

    my @timeline_down_temp  = ();
```

```
if($DEBUG == 1)
{
    # parse check

    print("src: $src \t");
    print("dest: $dst \t");
    print("path_number: $path_number \t");
    print("path: $path\t");
    print("up: $up\t");
    print("down $down\t");
    print("\n");
}

if($path_number=~ m/(0|1|2|3|4|5|6|7|8|9)/)
{
    # collect all source destination connections
    #-----------------------------------------#

    if($src_dest{$src}{$dst}=="")
    {
        # new connection

        # update total data connection counter
        $src_dest{"num con"}++;
        my $connection_index = $src_dest{"num con"};

        # save new connection in hash table
        $src_dest{$src}{$dst}           = $connection_index;
        $src_dest{$connection_index}{"src"} = $src;
        $src_dest{$connection_index}{"dst"} = $dst;

        printf("PARSE CONNECTION: NUM: %s \t SRC: %s \t DEST: %s\n",
                                    $connection_index, $src, $dst);
    }

    # collect all path (with a hash number)
    #--------------------------------------#

    if($src_dest_path{$src}{$dst}{$path}=="")
    {# new path

        $src_dest_path{$src}{$dst}{"num paths"}++;
        my $path_index  = $src_dest_path{$src}{$dst}{"num paths"};
        $src_dest_path{$src}{$dst}{$path_index}     = $path;
        $src_dest_path{$src}{$dst}{$path}           = $path_index;

        if($DEBUG == 1)
        {
            printf("\t\tPATH: NUM: %s \t PATH: %s \n",$path_index,$path);
        }
    }
    else
    {   #old path!
        die("import data: collect path: path already stored!! ERROR !!!");
    }

    # process up time events
    #-----------------------#

    # temporal array to construct time line
    my @timeline_up_temp    = ();

    # containers to work with up and down times
    my @uptimes = ();
    my $uptime  = 0;
```

```
# load timeline
#
if($timeline_up[$src][$dst] == "")
{#first path -> new
    @timeline_up_temp = ();
}
else
{# load old time line
    @timeline_up_temp = @{$timeline_up[$src][$dst]};
}

# load uptimes

@uptimes = split(/-/, $up); # split string

foreach $uptime (@uptimes) # for each up time
{
    if($uptime == "UP")
    { # header
        # do nothing
    }
    elsif($uptime =~ m/(0|1|2|3|4|5|6|7|8|9)/)
    {
        # save up_time in local time_line array
        $timeline_up_temp[$#timeline_up_temp + 1] = $uptime;

        # save event in event handler
        $event_hash{$src}{$dst}{"up"}{$uptime}{"num events"}++;
        $event_hash{$src}{$dst}{"up"}{$uptime}
            {$event_hash{$src}{$dst}{"up"}{$uptime}{"num events"}}
                =$path;
    }
    else
    {
        die("import_data: uptime parser: unknown $uptime");
    }
}

# save timeline
$timeline_up[$src][$dst] = [sort(number_sort @timeline_up_temp)];

# process down time events
#-------------------------#

# temporal array to construct time line
my @timeline_down_temp  = ();

# containers to work with up and down times
my @downtimes    = ();
my $downtime     = 0;

# load timeline
if($timeline_down[$src][$dst] == "")
{#first path --> new
    @timeline_down_temp = ();
}
else
{# load old time line
    @timeline_down_temp = @{$timeline_down[$src][$dst]};
}

@downtimes = split(/-/, $down);

foreach $downtime (@downtimes) # for each down time
{
    if($downtime == "DOWN")
    { #header
```

```
                # do nothing
            }
            elsif($downtime =~ m/(0|1|2|3|4|5|6|7|8|9)/)
            {
                # save down_time in local time_line array
                $timeline_down_temp[$#timeline_down_temp + 1] = $downtime;

                # save event in event handler
                $event_hash{$src}{$dst}{"down"}{$downtime}{"num events"}++;
                $event_hash{$src}{$dst}{"down"}{$downtime}
                    {$event_hash{$src}{$dst}{"down"}{$downtime}
                        {"num events"}} = $path;
            }
            else
            {
                die("import_data: downtime parser: unknown $downtime");
            }
        }
        # save timeline
        $timeline_down[$src][$dst] = [sort(number_sort @timeline_down_temp)];
    }
    else
    {
        die " RT: error... $line   \n";
    }
}
elsif ($elements[0] =~ "#RT-H-SRC")
{
    # Header Source table

}
elsif ($elements[0] =~ "#RT-SRC")
{   # rtable src
    # information of the routing table of the source of the data connection

    # interface definition
    my $src             = $elements[1];
    my $dst             = $elements[2];
    my $min_hop         = $elements[3];
    my $path_number_at_t_string = $elements[4];
    my $timeline_string = $elements[5];

    if($DEBUG == 1)
    {
        # parse check
        print("src: $src \t");
        print("dest: $dst \t");
        print("min hop: $min_hop \t");
        print("path: $path_number_at_t_string\t");
        print("timeline: $timeline_string\t");
        print("\n");
    }

    if($min_hop =~ m/(0|1|2|3|4|5|6|7|8|9)/)
    {# check that the line corresponds to a meaningful datasheet

        # number of active paths
        #-------------------------#

        # parse strings

        my @paths_num_at_t_array    = split(/-/, $path_number_at_t_string);
        my @timeline_array          = split(/-/, $timeline_string);

        #check if both array really have the same length

        if(scalar(@paths_num_at_t_array) != scalar(@timeline_array))
```

```
                {#have not the same length
                    die(" RT-SRC: Time-String has not the same length as".
                                " 'Number of paths'-String!!--> error\n");
                }

                # first element is only description
                $path_time_src{$src}{$dst}{"num slots"} = ( scalar(@timeline_array) - 1);

                #copy values in statistic container (start by slot zero)
                # start by one --> first element only description
                for(my $ii = 1; $ii<scalar(@paths_num_at_t_array); $ii++)
                {
                    $path_time_src{$src}{$dst}{$ii -1 }{"num paths"}
                            = $paths_num_at_t_array[$ii];

                    $path_time_src{$src}{$dst}{$ii -1 }{"time"}
                            = $timeline_array[$ii];
                }
                # min hop
                #--------------------
                $path_time_src{$src}{$dst}{"shortest path"} = $min_hop;
            }
        }
        else
        {
            # print ("no match", $elements[o] ,"\n");
        }
    }
    # normalisation (dropped packet)
    for(my $ii = o; $ii<$node_total;$ii++)
    {
        # Add +1 to be sure that we have no division by zero event ...
        $dropped_dpackets_v_tot_sent[$ii] =
                $dropped_packet_data_node[$ii]/(1+$sent_packet_data_total);

        $dropped_dpackets_v_received_node[$ii] =
            $dropped_packet_data_node[$ii]/(1+$received_packet_data_node[$ii]
                + $forwarded_packet_data_node[$ii]);
    }

    # close file
    close(SIM_FILE) or die "Couldn't close $sim_file\n";
}

sub number_sort
{
# helper function to sort time events
    if($a < $b)
    { return -1; }
    elsif($a == $b)
    { return o; }
    else
    { return 1; }
}

#-----# CALCULATIONS #-----#

sub calculate_stat_rt_s_t
{
    # calculations based on the information out of the routing table space time

    # calculate for $connection :
    # 1. find out which paths were active at which time
    # 2. calculate the number of found paths based on the routing table information
    # 3. calculate the average number of found paths based on the routing table info

    # Variable Declaration.
```

```
#constant
my $INFINITY        = 9999999;

#argument
my $connection      = shift(@_);

#local varables
my $src             = $src_dest{$connection}{"src"};
my $dst             = $src_dest{$connection}{"dst"};

my $path_total      = $src_dest_path{$src}{$dst}{"num paths"};

# Timeline for this connection
my @timeline_up_    = @{$timeline_up[$src][$dst]};
my @timeline_down_  = @{$timeline_down[$src][$dst]};

# pointer to the active element of the timeline
my $timeline_up_next_pointer    = 0;
my $timeline_down_next_pointer  = 0;

my $time                        = 0;
my $time_last                   = 0;

# Handle zero pointer:

if(scalar(@timeline_up_) == 0)
{ # no up event
    printf("no up event ...\n");
    $timeline_up_[0] = $INFINITY;
}

if(scalar(@timeline_down_) == 0)
{ # no down event
    printf("no down event ...\n");
    $timeline_down_[0] = $INFINITY;
}

my $time_up_next    = $timeline_up_[0];
my $time_down_next  = $timeline_down_[0];

# variable to save start of the connection
#my $time_connection_start  = $INFINITY;

# variable to save end of the connection
#my $time_connection_end        = 0;

# register to save the state of the paths during the time $time
my @path_active_register = ();
    # path_active_register[#path_hash] = $time_up

my $path_active_number      = 0; #number of active paths at time $time


my $event_type          = "";

# Initialisation of the Values
# (First Time Slot)
#-----------------------------------------#

if($time_up_next <= $time_down_next) # first event must be a UP-EVENT
{
    $time_last      = 0;
    $time           = $timeline_up_[0];

    # next up time
    while($timeline_up_next_pointer < scalar(@timeline_up_)
```

```
        && $timeline_up_[$timeline_up_next_pointer] == $time_up_next)
    {
        $timeline_up_next_pointer++;
    }
    $time_up_next       = $timeline_up_[$timeline_up_next_pointer];

    $time_down_next     = $timeline_down_[$timeline_down_next_pointer];
    $timeline_down_next_pointer++;

    # save first time
    $path_time_s_t{$src}{$dst}{"num slots"} = 1;
    $path_time_s_t{$src}{$dst}{1}{"time"}   = $time;

    # next event type
    $event_type             = "up";

    # number of paths
    $path_active_number     = 0;
    @path_active_register   = ();
}
else
{
    print("up_next: $time_up_next down_next: $time_down_next \n");
    die("CALC RT_S_T: first element in timeline was not a up event !!");
}


# Wake through the simulation
# until the end of the simulation
#-----------------------------------#

while($time < $INFINITY)
{
    # update data for $time
    #-----------------------#

    if($event_type =~ "up")
    {   #UP
        for(my $ii = 1;
                $ii <= $event_hash{$src}{$dst}{"up"}{$time}{"num events"};
                    $ii++)
        {
            # update active  path register
            my $loop_path   = $event_hash{$src}{$dst}{"up"}{$time}{$ii};
            my $loop_path_hash  = $src_dest_path{$src}{$dst}{$loop_path};
            $path_active_register [$loop_path_hash] = $time; # set path as active
        }
    }
    elsif($event_type =~ "down")
    {   #DOWN
        for(my $ii = 1;
                $ii <= $event_hash{$src}{$dst}{"down"}{$time}{"num events"};
                     $ii++)
        {
            # update active path register
            my $loop_path       = $event_hash{$src}{$dst}{"down"}{$time}{$ii};
            my $loop_path_hash  = $src_dest_path{$src}{$dst}{$loop_path};
            $path_active_register [$loop_path_hash] = 0; # set path as not working
        }
    }
    else
    {
        die("CALC RT_S_T: unknown event\n");
    }


    # calculate the number of active paths
```

```
$path_active_number = 0;
for(my $ii = 1; $ii <= $path_total; $ii++)
{
    if($path_active_register[$ii] > 0)
    {
        $path_active_number++;
    }
}

# save data
my $loop_slot = $path_time_s_t{$src}{$dst}{"num slots"};
$path_time_s_t{$src}{$dst}{$loop_slot}{"num active paths"}  = $path_active_number;
$path_time_s_t_active_paths[$src][$dst][$loop_slot] = [@path_active_register];


# update time
#--------------------#

my $decision = "";

# decision next event:
#   --> Set decision to UP/DOWN

if($time_up_next < $time_down_next)
{
    $decision = "UP";
}
elsif($time_up_next > $time_down_next)
{
    $decision = "DOWN";
}
elsif($time_up_next == $INFINITY && $time_down_next==$INFINITY)
{ # end of simulation, decision is not irrelevant
    $decision = "DOWN";
}
elsif($time_up_next == $time_down_next)
{
    # Problem: Time discretisation was too small.
    #           Events are probably collected in one time slot ...
    #
    # Solution: create independent events!!
    # for each up event --> ++
    # for each down event --> --

    my @action = ();
    for(my $ii = 1;
        $ii <= $event_hash{$src}{$dst}{"up"}{$time_up_next}{"num events"}; $ii++)
    {
        my $loop_path = $event_hash{$src}{$dst}{"up"}{$time_up_next}{$ii};
        my $loop_path_hash  = $src_dest_path{$src}{$dst}{$loop_path};

        $action[$loop_path_hash]++;
    }

    for(my $ii = 1;
    $ii <= $event_hash{$src}{$dst}{"down"}{$time_down_next}{"num events"}; $ii++)
    {
        my $loop_path = $event_hash{$src}{$dst}{"down"}{$time_down_next}{$ii};
        my $loop_path_hash  = $src_dest_path{$src}{$dst}{$loop_path};

        $action[$loop_path_hash]--;
    }

    # overwrite events
    # --> if  1 --> UP
    # --> if  0 --> nothing
    # --> if -1 --> DOWN
```

```
    # --> else --> ERROR!!!!

    $event_hash{$src}{$dst}{"up"}{$time_up_next}{"num events"} = 0;
    $event_hash{$src}{$dst}{"down"}{$time_down_next}{"num events"} = 0;

    for(my $ii = 1; $ii <= $path_total; $ii++)
    { # overwrite events

        my $loop_path = $src_dest_path{$src}{$dst}{$ii};

        if($action[$ii]==1)
        {# UP EVENT

            $event_hash{$src}{$dst}{"up"}{$time_up_next}{"num events"}++;

            $event_hash{$src}{$dst}{"up"}{$time_up_next}{$event_hash{$src}{$dst}
                {"up"}{$time_up_next}{"num events"}} = $loop_path;
        }
        elsif($action[$ii] == -1)
        {# DOWN EVENT

            $event_hash{$src}{$dst}{"down"}{$time_down_next}{"num events"}++;

            $event_hash{$src}{$dst}{"down"}{$time_down_next}{$event_hash{$src}
                {$dst}{"down"}{$time_down_next}{"num events"}} = $loop_path;
        }
        elsif($action[$ii] == 0)
        {# no action

        }
        else
        {# error !
            die("to many events for this path ... :$action[$ii]\n")
        }
    }
    $decision = "UP";
}

$time_last = $time;

# update time according to $decision
#-------------------------------------#

if($decision =~ "UP")
{   #Next event is a UP event

    $time       = $time_up_next;
    $event_type = "up";

    # calculate next up time
    while($timeline_up_next_pointer < scalar(@timeline_up_)
        && $timeline_up_[$timeline_up_next_pointer] == $time_up_next)
    {
        $timeline_up_next_pointer++;
    }
    if($timeline_up_next_pointer < scalar(@timeline_up_))
    {
        $time_up_next   = $timeline_up_[$timeline_up_next_pointer];
    }
    else
    {
        $time_up_next   = $INFINITY;
    }
}
elsif($decision =~ "DOWN")
{      # Next event is a DOWN event
```

```
            $time       = $time_down_next;
            $event_type = "down";

            # next down time
            while($timeline_down_next_pointer < scalar(@timeline_down_)
                && $timeline_down_[$timeline_down_next_pointer] == $time_down_next)
            {
                $timeline_down_next_pointer++;
            }
            if($timeline_down_next_pointer < scalar(@timeline_down_))
            {
                $time_down_next = $timeline_down_[$timeline_down_next_pointer];
            }
            else
            {
                $time_down_next = $INFINITY;
            }
        }
        else
        {
            die("Decision: $decision");
        }

        $path_time_s_t{$src}{$dst}{"num slots"}++;
        my $loop_time_slot = $path_time_s_t{$src}{$dst}{"num slots"};
        $path_time_s_t{$src}{$dst}{$loop_time_slot}{"time"} = $time;

    }#while end (END OF SIMULATION DATA)
}

sub calculate_stat_rt_src
{
    # calculate statistics based on the rtable_src (as src sees it)
    # 1. number of paths on average
    # 2. number of found paths for one route request

    # averages over a fix time-windows:
    #
    #   left window          right window
    #           Y                Y
    #
    #       ______|__________Â¡__
    #
    #           180          last data packet sent
    #

    my $connection          = shift(@_);
    my $INFINITY            = 9999999;

    my $src                 = $src_dest{$connection}{"src"};
    my $dst                 = $src_dest{$connection}{"dst"};

    my $shortest_path       = $path_time_src{$src}{$dst}{"shortest path"} ;

    my $time_last           = $path_time_src{$src}{$dst}{0}{"time"};
    my $time                = $path_time_src{$src}{$dst}{1}{"time"};
    my $number_of_paths     = $path_time_src{$src}{$dst}{0}{"num paths"};

    # Average 180 - Simulation End (last data packet)
    my $number_of_paths_time_180_avg        = 0;
    my $number_of_path_time_180_max         = 0;

    my $window_180_start                    = 180;
    my $window_180_stop                     = $simulation_end;


    # Routing Protocol Performance (Try to remove Interface Callback effect)
```

```
# Idea: find the number of paths that can be found by one route request
# Implementation: Route request --> Starts after a longer Zero Period
#                                   (no paths in the routing table)
#
#         Performance = Max Paths between to Zeros Periods
# Problem: Define longer Zero Period !!
#       - short zero events occurs due to routing table updates ...
#       - define 'Zero Periode' >> 'routing update'
#       - if a route request can't find a route --> longer zero period ..
#           --> this is not captured with this technique ...

my $ZERO_PERIOD_DURATION    = 0.01;  #[s]
my $locale_maximum          = 0;

#-----------------------------------
# First loop over the simulation
#-----------------------------------

# calc number of found paths on average
# calc number of found paths for one route request

for(my $ii = 1; $ii < $path_time_src{$src}{$dst}{"num slots"}; $ii++)
{
    # Routing Protocol Performance
    #-------------------------------#
    if($time_last > 180)
    {
        #find local maximum
        if($number_of_paths > $locale_maximum)
        {
            $locale_maximum = $number_of_paths;
        }

        #check if this is a 'Zero Period'
        if(($time-$time_last >= $ZERO_PERIOD_DURATION )&& ($number_of_paths == 0))
        {
            $rp_performace[$shortest_path][$locale_maximum]++;
            $locale_maximum = 0;
        }
    }

    # Average 180
    #-------------#

    if($time > $window_180_start && $time_last < $window_180_stop)
    {
        # check window

        my $left_side = 0;
        my $right_side = 0;

        if($window_180_start > $time_last )
        {
            $left_side  = $window_180_start;
        }
        else
        {
            $left_side  = $time_last;
        }

        if($window_180_stop < $time)
        {
            $right_side = $window_180_stop;
        }
        else
        {
            $right_side = $time;
```

```
            }

            # Time Average
            #---------------#

            $number_of_paths_time_180_avg += ($right_side - $left_side)*$number_of_paths;

            if(0)
            {
                printf("DEBUG calc src: SRC %s DEST %s \n TIME %s LAST TIME %s".
                    " dT %s NUM Path %s LEFT %s RIGHT%s AVG_NOT_NORM %s\n",
                    $src, $dst, $time, $time_last, ($time - $time_last), $number_of_paths,
                    $left_side, $right_side, $number_of_paths_time_180_avg);
                my $menu = <STDIN>;
                chomp $menu;
            }

            # Max Paths
            #--------------#
            if($number_of_paths > $number_of_path_time_180_max)
            {
                $number_of_path_time_180_max = $number_of_paths;
            }
        }


        #update data ($time)
        $number_of_paths    = $path_time_src{$src}{$dst}{$ii}{"num paths"};

        # next time
        $time_last          = $time;
        $time               = $path_time_src{$src}{$dst}{$ii + 1}{"time"};
    }

    # EXCEPTION HANDLE OF AVERAGE CALCULATION
    #
    # Problem
    #   1: if no event occurs during the window size --> adjust left and right window side
    #   2: if there is no event outside the windows --> adjust right window side

    if($time_last < $window_180_stop)
    {
        my $left_side;
        my $right_side;

        $right_side = $window_180_stop;

        if($time_last < $window_180_start)
        {
            $left_side = $window_180_start;
        }
        else
        {
            $left_side  = $time_last;
        }

        $number_of_paths_time_180_avg += ($right_side - $left_side) * $number_of_paths;

        if(0)
        {
        printf("DEBUG calc src: SRC %s DEST %s \n".
            " TIME %s LAST TIME %s dT %s NUM Path %s LEFT %s RIGHT%s AVG_NOT_NORM %s\n",
                $src, $dst, $time, $time_last, ($time - $time_last), $number_of_paths,
                $left_side, $right_side, $number_of_paths_time_180_avg);
        }
    }
```

```
    # Normalisation (time average) (plus 1 ==> avoid division by zero)

    $number_of_paths_time_180_avg   =
            $number_of_paths_time_180_avg / ($window_180_stop - $window_180_start + 1);

    $path_time_src{$src}{$dst}{"180 avg"}   = $number_of_paths_time_180_avg;

    # max path
    $path_time_src{$src}{$dst}{"180 max"}   = $number_of_path_time_180_max;

    printf ("\n\nCALC_SRC: SRC %s DEST %s TIME %s \n", $src,$dst,$time);
    printf ("CALC_SRC 180 mean: %s max: %s left %s right %s time: %s last time: %s\n\n",
        $path_time_src{$src}{$dst}{"180 avg"}, $path_time_src{$src}{$dst}{"180 max"},
        $window_180_start, $window_180_stop, $time, $time_last);
}

sub calc_stat_shortest
{
    # calculate all statistics in respect on 'shortest path'

    calc_stat_shortest_connection();
    calc_stat_shortest_180_time();
    calc_stat_shortest_180_max();
    calc_stat_shortest_180_rp_p();
}

sub calc_stat_shortest_connection
{
    # calculate the distribution of shortest path lengths in the simulation
    # find the longest 'shortest path' in the simulation

    my $number_of_connections = $src_dest{"num con"};

    my @connections = ();
    my $shortest_path_max = 0;

    # loop variables
    my $src      = 0;
    my $dst      = 0;
    my $shortest_path = 0;


    for(my $ii = 1; $ii <= $number_of_connections; $ii++) #loop over con
    {
        # interface
        $src            = $src_dest{$ii}{"src"};
        $dst            = $src_dest{$ii}{"dst"};
        $shortest_path  = $path_time_src{$src}{$dst}{"shortest path"};

        # distribution of the connections
        $connections[$shortest_path]++;

        # longest 'shortest path'
        if($shortest_path_max < $shortest_path)
        {
            $shortest_path_max = $shortest_path;
        }
    }

    # save data
    for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop over shortest
    {
        if($connections[$ii] == "") # problem by converting "" to int
        {
            $path_shortest{$ii}{"num con"} = 0;
        }
        else
```

```
        {
            $path_shortest{$ii}{"num con"} = $connections[$ii];
        }
    }

    $path_shortest{"shortest path max"} = $shortest_path_max;

    #DEBUG OUTPUT
    if($DEBUG == 1)
    {
        my $connection_sum = 0;

        printf("\n\n -- calc_stat_connection --\n");
        printf("Connection Statistic\n");
        printf("max shortest path: %i\n",$path_shortest{"shortest path max"});

        for(my $ii=1;$ii <= $path_shortest{"shortest path max"};$ii++)
        {
            printf("Shortest path: %2.i number of connections: %4.i\n",
                $ii, $path_shortest{$ii}{"num con"});

            $connection_sum += $path_shortest{$ii}{"num con"};
        }

        #check if some connections were lost in space ...
        if($connection_sum!=$src_dest{"num con"})
        {
            die " calc_stat_connection: Difference in total number of connection ...";
        }
        else
        {
            printf("Number of Connections: %i\n",$src_dest{"num con"});
        }
    }
}

sub calc_stat_shortest_180_time
{
    # TIME AVERAGE
    # Calculate the number of active paths in time average
    # Start time = 180
    # End time = time when the last data packet was sent

    # INPUT:

    # OUTPUT:
    #   $path_shortest{$ii}{"180 time avg"} = $avg;
    #   $path_shortest{$ii}{"180 time var"} = $var;


    # calculate statistics with respect to the shortest path:

    my $number_of_connections   = $src_dest{"num con"};
    my $shortest_path_max       = $path_shortest{"shortest path max"};

    # interface
    my $src_sample      = 0;
    my $dst_sample      = 0;
    my $avg_sample      = 0;
    my $shortest_path_sample = 0;

    # local data container
    my @avg = ();       # average
    my @var = ();       # variance
    my @samples = ();   # number of connections =~ samples

    # AVERAGE
```

```
#----------#

for(my $ii = 1; $ii <= $number_of_connections; $ii++) # loop over con
{
    # interface
    $src_sample = $src_dest{$ii}{"src"};
    $dst_sample = $src_dest{$ii}{"dst"};

    $avg_sample      = $path_time_src{$src_sample}{$dst_sample}{"180 avg"};
    $shortest_path_sample= $path_time_src{$src_sample}{$dst_sample}{"shortest path"};

    # calc
    $avg[$shortest_path_sample] += $avg_sample;
    $samples[$shortest_path_sample]++;
}

# normalisation
for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop over shortest
{
    if( $samples[$ii] > 0) # division by zero
    {
        $avg[$ii] = $avg[$ii]/ $samples[$ii];
    }
    else
    {
        $avg[$ii] = 0;
    }
}

# VARIANCE
#----------#

for(my $ii = 1; $ii <= $number_of_connections; $ii++) # loop over con
{
    # interface
    $src_sample      = $src_dest{$ii}{"src"};
    $dst_sample      = $src_dest{$ii}{"dst"};

    $avg_sample      = $path_time_src{$src_sample}{$dst_sample}{"180 avg"};
    $shortest_path_sample = $path_time_src{$src_sample}{$dst_sample}{"shortest path"};

    # calc
    $var[$shortest_path_sample] += ($avg_sample - $avg[$shortest_path_sample] )**2;
}

# normalisation
for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop over shortest
{
    if($samples[$ii] > 0) # division by zero
    {
        $var[$ii] = $var[$ii] / $samples[$ii];
    }
    else
    {
        $var[$ii] = 0;
    }
}

# SAVE DATA
#----------#

for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop shortest path
{
    $path_shortest{$ii}{"180 time avg"} = $avg[$ii];
    $path_shortest{$ii}{"180 time var"} = $var[$ii];
}
```

```
    # DEBUG OUTPUT
    if($DEBUG == 1)
    {
        printf("\n\n -- calc_stat_180_time --\n");
        printf("Active paths over time over all connections\n");
        for(my $ii = 1; $ii <= $shortest_path_max; $ii++)
        {
            printf("Shortest path: %2.i avg: %2.4f var: %2.4f\n", $ii,
                $path_shortest{$ii}{"180 time avg"},
                $path_shortest{$ii}{"180 time var"});
        }
    }
}

sub calc_stat_shortest_180_max
{
    # MAXIMUM OVER TIME
    # Calculate the number of maximum active paths in time average over all connections

    # OUTPUT:
    #   $path_shortest{$ii}{"180 max avg"} = $avg;
    #   $path_shortest{$ii}{"180 max var"} = $var;


    # calculate statistics with respect to the shortest path:

    my $number_of_connections   = $src_dest{"num con"};
    my $shortest_path_max       = $path_shortest{"shortest path max"};

    # interface
    my $src_sample      = 0;
    my $dst_sample      = 0;
    my $max_sample      = 0;
    my $shortest_path_sample= 0;

    # local data container
    my @avg = ();       # average
    my @var = ();       # variance
    my @samples = ();   # number of connections =~ samples

    # AVERAGE
    #----------#

    for(my $ii = 1; $ii <= $number_of_connections; $ii++) # loop over connections
    {
        # interface
        $src_sample     = $src_dest{$ii}{"src"};
        $dst_sample     = $src_dest{$ii}{"dst"};

        $max_sample     = $path_time_src{$src_sample}{$dst_sample}{"180 max"};
        $shortest_path_sample= $path_time_src{$src_sample}{$dst_sample}{"shortest path"};

        # calc
        $avg[$shortest_path_sample] += $max_sample;
        $samples[$shortest_path_sample]++;
    }

    # normalisation
    for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop over shortest path
    {
        if( $samples[$ii] > 0) # division by zero
        {
            $avg[$ii] = $avg[$ii]/ $samples[$ii];
        }
        else
        {
            $avg[$ii] = 0;
```

```
        }
    }

    # VARIANCE
    #----------#

    for(my $ii = 1; $ii <= $number_of_connections; $ii++) # loop over con
    {
        # interface
        $src_sample      = $src_dest{$ii}{"src"};
        $dst_sample      = $src_dest{$ii}{"dst"};

        $max_sample      = $path_time_src{$src_sample}{$dst_sample}{"180 max"};
        $shortest_path_sample = $path_time_src{$src_sample}{$dst_sample}{"shortest path"};

        # calc
        $var[$shortest_path_sample] += ($max_sample - $avg[$shortest_path_sample] )**2;
    }

    # normalisation
    for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop over shortest path
    {
        if($samples[$ii] > 0) # division by zero
        {
            $var[$ii] = $var[$ii] / $samples[$ii];
        }
        else
        {
            $var[$ii] = 0;
        }
    }

    # SAVE DATA
    #----------#

    for(my $ii = 1; $ii <= $shortest_path_max; $ii++) # loop shortest path
    {
        $path_shortest{$ii}{"180 max avg"} = $avg[$ii];
        $path_shortest{$ii}{"180 max var"} = $var[$ii];
    }

    # DEBUG OUTPUT
    if($DEBUG == 1)
    {
        printf("\n\n -- calc_stat_180_max --\n");
        printf("Maximum Active paths over time over all connections\n");
        for(my $ii = 1; $ii <= $shortest_path_max; $ii++)
        {
            printf("Shortest path: %2.i avg: %2.4f var: %2.4f\n",
                $ii,
                $path_shortest{$ii}{"180 max avg"},
                $path_shortest{$ii}{"180 max var"});

        }
    }
}

sub calc_stat_shortest_180_rp_p
{
    # Calculate the Performance of the Routing Protocol
    # by calculating the average found paths by one Route Request
    # INPUT: $rp_performance array
    # OUTPUT:
    #   $path_shortest{#shortest path}{"180 rp_p avg"}
    #   $path_shortest{#shortest path}{"180 rp_p var}
    #   $path_shortest{#shortest path}{"180 rp_p samples"}
```

```
#Bound Array
#   worst case ...
my $LONGEST_SHORTEST_PATH = $path_shortest{"shortest path max"};
my $MAX_PATHS = 100;

for(my $ii = 1; $ii<= $LONGEST_SHORTEST_PATH; $ii++) #shortest path
{
    my $avg = 0;
    my $var = 0;
    my $route_request_total =0;

    # average & number of requests
    for(my $jj = 1; $jj <= $MAX_PATHS; $jj++)
    {
        $avg += $jj*$rp_performace[$ii][$jj];
        $route_request_total += $rp_performace[$ii][$jj];
    }

    if($route_request_total > 0) # division by zero
    {
        $avg = $avg / $route_request_total;
    }
    else
    {
        $avg = 0;
    }


    # variance
    for(my $jj = 1; $jj <= $MAX_PATHS; $jj++)
    {
        $var += $rp_performace[$ii][$jj]*(($jj - $avg)**2);
    }

    if($route_request_total>0) #division by zero
    {
        $var = $var / $route_request_total;
    }
    else
    {
        $var = 0;
    }

    # save data in statistic container

    $path_shortest{$ii}{"180 rp_p avg"} = $avg;
    $path_shortest{$ii}{"180 rp_p var"} = $var;
    $path_shortest{$ii}{"180 rp_p num rreq"} = $route_request_total;

}
#DEBUG OUTPUT
if($DEBUG == 1)
{
    my $sample_total = 0;

    printf("\n\n -- calc_stat_180_rp_p --\n");
    printf("Routing Protocol Performance\n");
    for(my $ii = 0; $ii <= $path_shortest{"shortest path max"}; $ii++)
    {
        printf("Shortest path: %4.i avg: %6.3f var: %6.3f samples: %4.i\n", $ii,
            $path_shortest{$ii}{"180 rp_p avg"},
            $path_shortest{$ii}{"180 rp_p var"},
            $path_shortest{$ii}{"180 rp_p num rreq"});

        $sample_total += $path_shortest{$ii}{"180 rp_p num rreq"};
    }
```

```
        printf("Total number of samples: %i \n",$sample_total);

        # show samples

        # Header
        printf("\n");
        printf("Number of samples (shortest path – found paths\n");

        printf("Paths:\t 00\t 01 \t 02 \t 03 \t 04 \t 05 \t 06 \t 07 \t 08 \t 09\n");
        printf("————————————————————————————————————————————————————————————\n");

        for(my $ii = 1; $ii <= $path_shortest{"shortest path max"} + 2; $ii++)
        {
            printf("SP: %4.i | \t ",$ii);
            for(my $jj = 0; $jj <= 10; $jj++) #active paths
            {
                printf("%2.i\t ", $rp_performace[$ii][$jj]);
            }
            printf("\n");
        }
        printf(" column 00 —–> double zero interval ... ERRORS ... \n");
    }
}

sub connection_info
{
    # User interface to plot different data for one connection

    my $connection              = shift(@_);

    my $src                     =   $src_dest{$connection}{"src"};
    my $dst                     =   $src_dest{$connection}{"dst"};

    # rtable space time
    my $path_total              = $src_dest_path{$src}{$dst}{"num paths"};
    my @timeline_up_temp        = @{$timeline_up[$src][$dst]};
    my @timeline_down_temp      = @{$timeline_down[$src][$dst]};

    my $time_s_t_slots_total    = $path_time_s_t{$src}{$dst}{"num slots"};

    # rtable source
    my $path_180_ave            = $path_time_src{$src}{$dst}{"180 time ave"};
    my $path_180_max            = $path_time_src{$src}{$dst}{"180 time max"};
    my $path_shortest           = $path_time_src{$src}{$dst}{"shortest path"};


    my $menu = 1; # control menu
    while($menu)
    {
        printf("\n\n");
        printf(" #————————————————————————————————————————————————————————# \n");
        printf(" # INFO FOR CONNECTION %s                                      # \n",$connection);
        printf(" #————————————————————————————————————————————————————————# \n");
        printf(" \n\n");
        printf("SRC: %s  \t   DEST: %s \n",$src, $dst);
        printf("SHORTEST PATH :   %s\n",$path_shortest);
        printf("PATHS TOTAL:      %s\n",$path_total);
        printf("ACTIVE #PATH AVG: %s\n",$path_180_ave);
        printf("ACTIVE #PATH MAX: %s\n",$path_180_max);
        print "\n";
        print "\n";
        print "\n";
        print "Path–Time PLOT                                    1\n";
        print "Path–Time TEXT                                    2\n";
        print "Found paths in routing table PLOT                 3\n";
        print "Make animated GIF                                 4\n";
```

```
print "DEBUG: Make animated GIF small                    5\n";
print "DEBUG: List all paths                             6\n";
print "DEBUG: List all UP-EVENTS                         7\n";
print "DEBUG: List all DOWN_EVENTS                       8\n";
print "\n";
print "RETURN                                            0\n";

$menu = <STDIN>;
chomp $menu;

if($menu eq 1)
{# PLOT: Path-Time
    plot_path_time($connection);
}
elsif($menu eq 2)
{# TEXT: Path-Time
    show_path($connection);
}
elsif($menu eq 3)
{ #PLOT: active paths from time $start to time $end

    # save eps
    my $folder = "temp";
    my $filename = "non";

    my $start = 1;
    my $end  = 0;

    # show paths over time a-b
    while ($start > 0)
    {
        printf("\nPlot all found paths in the routing ".
                "table over a given time interval \n");
        printf("RETURN => 0\n");
        printf("\n");

        printf("first time slot: 1 - %s \n",$time_s_t_slots_total-1);
        $start = <STDIN>;
        chomp $start;

        printf("last time slot: %s - %s \n",$start,$time_s_t_slots_total-1);
        $end = <STDIN>;
        chomp $end;


        if($start > 0 && $end <= $time_s_t_slots_total) # Input validation
        {
            # find all paths that were active during this interval

            my @active_path_interval    =();
            my @active_path             =();

            # construct path list
            for(my $slot = $start; $slot <= $end; $slot++) # over time
            {
                @active_path = @{$path_time_s_t_active_paths[$src][$dst][$slot]};

                for(my $path = 1; $path <= $path_total; $path++) # over paths
                {
                    if($active_path[$path] > 0)
                    {
                        @active_path_interval[$path] =1;
                    }
                }
            }
            # generate plot
            $filename = sprintf ("%s/CON-%s-SRC-%s-DEST-%s-TIME-START-%i-END-%i",
```

```
                        $folder,$connection, $src, $dst, $start, $end);

            plot_path_space($connection,"$filename.eps",
                        [@active_path_interval],"SLOT: $start – $end");

            # view plot
            print `gv $filename.eps`;
        }
    }
}
elsif($menu eq 4)
{ #make animated gif
    printf("Make Animated Gif....\n");
    make_movie($connection);
}
elsif($menu eq 5)
{ #make small animated gif
    printf("Make Animated Gif small....\n");

    printf("START SLOT: max%s\n",$time_s_t_slots_total-1);
    my $start = <STDIN>;
    chomp $start;

    printf("END SLOT: max %s\n",$time_s_t_slots_total-1);
    my $end = <STDIN>;
    chomp $end;

    make_movie_short($connection,$start,$end);
}
elsif($menu eq 6)
{ #list all paths
    printf("Paths: \n");

    for(my $ii = 1; $ii <= $path_total; $ii++)
    {
        printf("\t  NR: %i \t PATH: %s\n",
                $ii, $src_dest_path{$src}{$dst}{$ii});
    }
}
elsif($menu eq 7)
{ #list all up events

    printf("UP Events: %i \n",scalar(@timeline_up_temp));

    for(my $ii = 0; $ii < scalar(@timeline_up_temp); $ii++)
    {
        printf( "\t time: %f \t number of events %i \t \n",
                $timeline_up_temp[$ii], $event_hash{$src}{$dst}{"up"}
                {$timeline_up_temp[$ii]}{"num events"} );
    }
}

elsif($menu eq 8)
{ #list all down events
    printf("DOWN Events: %i \n",scalar(@timeline_down_temp));

    for(my $ii = 0; $ii < scalar(@timeline_down_temp); $ii++)
    {
        printf( "\t time: %f \t number of events %i \t \n",
            $timeline_down_temp[$ii],
            $event_hash{$src}{$dst}{"down"}
            {$timeline_down_temp[$ii]}{"num events"});
    }
}
elsif($menu eq 0)
{ #return
```

```
        }

        else
        {
            print "unknown menu: $menu";

        }
    }
    printf("-----------------------------------\n");
    printf("         END - INFO - END          \n");
    printf("-----------------------------------\n");
}

sub show_path
{
    # list all path at time x

    printf("show path\n");
    # extract paths at time x and display them in ASCII

    my $connection      = shift(@_);

    my $src         = $src_dest{$connection}{"src"};
    my $dst         = $src_dest{$connection}{"dst"};

    my $path_total  = $src_dest_path{$src}{$dst}{"num paths"};

    my $time        = 0;

    my $time_slots_total_s_t = $path_time_s_t{$src}{$dst}{"num slots"};
    my @active_path     =();

    printf("show path for: SRC %s DEST %s  TOTAL-PATHS %s  NUM-TIMESLOTS %s\n",
            $src, $dst, $path_total, $time_slots_total_s_t);

    for(my $ii = 1; $ii < $time_slots_total_s_t; $ii++)
    {#update data
        $time           = $path_time_s_t{$src}{$dst}{$ii}{"time"};
        @active_path    = @{$path_time_s_t_active_paths[$src][$dst][$ii]};

        printf("\nTIME: %s \n", $time);

        for( my $jj = 0; $jj <= $path_total; $jj++)
        {
            if($active_path[$jj] > 0)
            {
                printf("\t\tuptime %s relative %s path (%s) %s \n",
                        $active_path[$jj], $time - $active_path[$jj],
                        $jj, $src_dest_path{$src}{$dst}{$jj});
            }
        }
    }
}

sub make_movie()
{
#   make an animated gif out of the path space time info for one connection
#   animation is based on whirlgif (http://www.danbbs.dk/~dino/whirlgif/)

    my $connection              = shift(@_);
    my $src                     = $src_dest{$connection}{"src"};
    my $dst                     = $src_dest{$connection}{"dst"};
    my $path_total              = $src_dest_path{$src}{$dst}{"num paths"};

    my $time_s_t_slots_total    = $path_time_s_t{$src}{$dst}{"num slots"};

    # folder where the movie is saved
```

```
my $filename_movie          = "animated-SRC-$src-DEST-$dst";
my $folder                  = "movie";
mkdir $folder;
my $whirl_arg               = ""; # argument for whirl
my $TIME_SCALE_FACTOR       = 10;

# variables needed to generate the plots
my @active_path             = ();
my $time                    = 0;
my $time_next               = 0;
my $slot                    = 0;
my $filename                = "";

# data structure to remember the filename and time to generate the movie
my @filename_container      = ();
my @time_container          = ();
my @delay_container         = ();

# DEBUG OUT
printf("\n\n ---- make movie ----\n");

# Generate eps
#----------------#

# time zero
for(my $ii = 0; $ii <= $path_total; $ii++)
{
    $active_path[$ii] = 0;
}
$slot               = 0;
$time               = 0;
$time_next          = $path_time_s_t{$src}{$dst}{$slot+1}{"time"};
$time_container[0]  = $time;
$delay_container[0] = int(($time_next - $time)* $TIME_SCALE_FACTOR);


$filename    = sprintf ("%s/CON-%s-SRC-%s-DEST-%s-TIME-%012.6f",
                $folder, $connection, $src, $dst, $time);

$filename_container[0] = $filename;

plot_path_space($connection, "$filename.eps", [@active_path],
        "TIME: $time - $time_next  IMAGE $slot of $time_s_t_slots_total");

# time 1 to infinity
#start: slot = 1 end: slot =  $time_s_t_slots_total-1
for($slot = 1; $slot < $time_s_t_slots_total; $slot++)
{
    # update data for time slot $slot
    $time_next              = $path_time_s_t{$src}{$dst}{$slot+1}{"time"};
    $time                   = $path_time_s_t{$src}{$dst}{$slot}{"time"};

    $time_container[$slot]  = $time;

    $delay_container[$slot] = int(($time_next - $time)* $TIME_SCALE_FACTOR);


    $filename = sprintf ("%s/CON-%s-SRC-%s-DEST-%s-TIME-%012.6f",
                $folder, $connection, $src, $dst, $time);

    $filename_container[$slot] = $filename;

    @active_path = @{$path_time_s_t_active_paths[$src][$dst][$slot]};

    # create plot

    plot_path_space($connection,"$filename.eps",[@active_path],
```

```
            "TIME: $time – $time_next IMAGE $slot of $time_s_t_slots_total");


        if($DEBUG == 1)
        {
            printf("SLOT %4.i TIME %12.6f DELAY ABS %12.6f  ".
                "DELAY SCALE(1/100s) %12.6f  FILENAME: %s\n",
                $slot, $time, ($time_next – $time), $delay_container[$slot], $filename);
        }
        # view plot
        #print 'gv $filename.eps';
    }

    # Generate Movie
    # #———————————#

    # We used the 'whirlgif' to create our movie

    # Use 'convert' of the ImageMagick Packet to convert eps to gif
    #   – o <outputfile>
    #   – t <delay> in 1/100 s

    for (my $ii = 0; $ii<$time_s_t_slots_total; $ii++)
    {
        print "convert img $ii of $time_s_t_slots_total\n";

        # convert eps to gif
        my $command = "convert –density 200 $filename_container[$ii].eps".
                            " $filename_container[$ii].gif";

        print '$command \n';

        # crop gif
        $command = "convert –crop 640x600+200+80! $filename_container[$ii].gif".
                        " $filename_container[$ii].gif";

        print '$command';

        # create the argument string for whirl
        $whirl_arg =
            "$whirl_arg –time $delay_container[$ii] $filename_container[$ii].gif";
    }

    # create the animated gif
    print "create animated gif: $filename_movie.gif \n";
    print '$whirlgif –o $filename_movie.gif $whirl_arg';
}

sub make_movie_short()
{
# DEBUG
# like make_movie but reduce size of gif ...

    my $connection      = shift(@_);
    my $slot_start      = shift(@_);
    my $slot_end        = shift(@_);

    my $src             = $src_dest{$connection}{"src"};
    my $dst             = $src_dest{$connection}{"dst"};
    my $path_total      = $src_dest_path{$src}{$dst}{"num paths"};

    my $time_s_t_slots_total    = $path_time_s_t{$src}{$dst}{"num slots"};

    # folder where the movie is saved
    my $filename_movie      = "animatedgif–short–SRC–$src–DEST–$dst";
    my $folder              = "movie";
    mkdir $folder;
```

```
my $whirl_arg           = "";
my $TIME_SCALE_FACTOR   = 10;

# variables needed to generate the plots
my @active_path         = ();
my $time                = 0;
my $time_next           = 0;
my $slot                = 0;
my $filename            = "";

# data structure to remember the filename and time to generate the movie
my @filename_container  =();
my @time_container      =();
my @delay_container     =();

# DEBUG OUT
printf("\n\n ---- make movie ----\n");

# Generate eps
#--------------#

if($slot_start == 0)
{
    # time zero
    for(my $ii = 0; $ii <= $path_total; $ii++)
    {
        $active_path[$ii] = 0;
    }
    $slot               = 0;
    $time               = 0;
    $time_next          = $path_time_s_t{$src}{$dst}{$slot+1}{"time"};
    $time_container[0]  = $time;
    $delay_container[0] = int(($time_next - $time)* $TIME_SCALE_FACTOR);


    $filename   = sprintf ("%s/CON-%s-SRC-%s-DEST-%s-TIME-%012.6f",
                    $folder, $connection, $src, $dst, $time);

    $filename_container[0] = $filename;

    plot_path_space($connection,"$filename.eps",[@active_path],
            "TIME: $time - $time_next");
    $slot = 1;
}
else
{
    $slot = $slot_start;
}

# time 1 to infinity
#start: slot = 1 end: slot =  $time_s_t_slots_total-1
for(; $slot < $slot_end; $slot++)
{
    # update data for time slot $slot
    $time_next      = $path_time_s_t{$src}{$dst}{$slot+1}{"time"};
    $time           = $path_time_s_t{$src}{$dst}{$slot}{"time"};
    $delay_container[$slot] = int(($time_next - $time)* $TIME_SCALE_FACTOR);
    $time_container[$slot] = $time;

    $filename   = sprintf ("%s/CON-%s-SRC-%s-DEST-%s-TIME-%012.6f",
            $folder, $connection, $src, $dst, $time);
    $filename_container[$slot] = $filename;

    @active_path    = @{$path_time_s_t_active_paths[$src][$dst][$slot]};

    # create plot
```

```
        plot_path_space($connection,"$filename.eps",[@active_path],
                "TIME: $time – $time_next");

        if($DEBUG == 1)
        {
            printf("SLOT %4.i TIME %12.6f DELAY ABS %12.6f  ".
                "DELAY SCALE(1/100s) %12.6f  FILENAME: %s\n",
                $slot, $time, ($time_next – $time), $delay_container[$slot], $filename);
        }
        # view plot
        #print 'gv $filename.eps';
    }

    # Generate Movie
    #————————————#

    # We used the 'whirlgif' to create our movie

    # Use 'convert' of the ImageMagick Packet to convert eps to gif
    #   – o <outputfile>
    #   – t <delay> in 1/100 s

    for (my $ii = $slot_start; $ii < $slot_end; $ii++)
    {
        if($delay_container[$ii] > 5) #process only visible changes
        {
            print "convert img ($filename_container[$ii]) $ii of $time_s_t_slots_total\n";

            # convert eps to gif
            my $command ="convert –density 200 $filename_container[$ii].eps".
                    " $filename_container[$ii].gif";
            print '$command';

            # crop gif

            $command ="convert –crop 640x600+200+80! $filename_container[$ii].gif".
                        " $filename_container[$ii].gif";

            print '$command';

            # create the argument string for whirl
            $whirl_arg = "$whirl_arg –time $delay_container[$ii] ".
                            "$filename_container[$ii].gif";
        }
    }
    # create the animated gif
    print "create animated gif: $filename_movie.gif \n";
    print '$whirlgif –o $filename_movie.gif $whirl_arg';
}

sub plot_path_time
{
#   plot number of paths over the time

    my $connection              = shift(@_);

    my $src                     = $src_dest{$connection}{"src"};
    my $dst                     = $src_dest{$connection}{"dst"};

    my $INFINITY                = 99999999;

    # rtable space time
    my $s_t_time                = 0;
    my $s_t_time_slot_total     = $path_time_s_t{$src}{$dst}{"num slots"};
    my $s_t_active_paths        = 0;
    my $s_t_average_paths       = 0; #$path_time_s_t{$src}{$dst}{"time average"};
```

```
# rtable src
my $src_time                = 0;
my $src_time_slot_total     = $path_time_src{$src}{$dst}{"num slots"};
my $src_active_paths        = 0;
my $src_average_paths_180   = $path_time_src{$src}{$dst}{"180 avg"};

# gnuplot
my $data_file_s_t           = "gnuplot_data_paths_time_s_t.tmp";
my $data_file_src           = "gnuplot_data_paths_time_src.tmp";
my $header_file             = "gnuplot_header.tmp";
my $output_file             = "$src-$dst-path-time.eps";
my $max_slots               = 0;

#write data file
print "generate gnu data file ... \n";

#rtable_s_t
open(GNUPLOT_DATA,">$folder_temp/$data_file_s_t") ||
die "cant write gnuplot_data.tmp file";

for(my $ii = 0; $ii < $s_t_time_slot_total; $ii++)
{
    # time update
    $s_t_time           = $path_time_s_t{$src}{$dst}{$ii}{"time"};

    # plot old data an new time: (datapoint 1)

    # 01: rtable_s_t:  time
    print GNUPLOT_DATA ($s_t_time,"\t");

    # 02: rtable_s_t:  number of active paths
    print GNUPLOT_DATA ($s_t_active_paths,"\t");

    # 03: rtable_s_t:  number of paths on average
    print GNUPLOT_DATA ($s_t_average_paths,"\t");
    print GNUPLOT_DATA ("\n");

    #update data
    $s_t_active_paths   = $path_time_s_t{$src}{$dst}{$ii}{"num active paths"};

    # plot new data at new time: (datapoint 2)

    # 01: rtable_s_t:  time
    print GNUPLOT_DATA ($s_t_time,"\t");

    # 02: rtable_s_t:  number of active paths
    print GNUPLOT_DATA ($s_t_active_paths,"\t");

    # 03: rtable_s_t:  number of paths on average
    print GNUPLOT_DATA ($s_t_average_paths,"\t");
    print GNUPLOT_DATA ("\n");
}

# last point in the infinity

# 01: rtable_s_t:  time
print GNUPLOT_DATA ($INFINITY,"\t");

# 02: rtable_s_t:  number of active paths
print GNUPLOT_DATA ($s_t_active_paths,"\t");

# 03: rtable_s_t:  number of paths on average
print GNUPLOT_DATA ($s_t_average_paths,"\t");

close(GNUPLOT_DATA);

#rtable_src
```

```
open(GNUPLOT_DATA,">$folder_temp/$data_file_src") ||
die "cant write gnuplot_data.tmp file";

for(my $ii = 0; $ii < $src_time_slot_total; $ii++)
{
    # time update
    $src_time = $path_time_src{$src}{$dst}{$ii}{"time"};

    # plot old data an new time: (datapoint 1)

    # 01: rtable_src:  time
    print GNUPLOT_DATA ($src_time,"\t");

    # 02: rtable_src:  number of active paths
    print GNUPLOT_DATA ($src_active_paths,"\t");

    # 03: rtable_src:  number of paths on average
    print GNUPLOT_DATA ($src_average_paths_180,"\t");
    print GNUPLOT_DATA ("\n");

    #update data
    $src_active_paths = $path_time_src{$src}{$dst}{$ii}{"num paths"};

    # plot new data at new time: (datapoint 2)

    # 01: rtable_src:  time
    print GNUPLOT_DATA ($src_time,"\t");

    # 02: rtable_src:  number of active paths
    print GNUPLOT_DATA ($src_active_paths,"\t");

    # 03: rtable_src:  number of paths on average
    print GNUPLOT_DATA ($src_average_paths_180,"\t");
    print GNUPLOT_DATA ("\n");
}

# last point in the infinity

# 01: rtable_src:  time
print GNUPLOT_DATA ($INFINITY,"\t");

# 02: rtable_src:  number of active paths
print GNUPLOT_DATA ($src_active_paths,"\t");

# 03: rtable_src:  number of paths on average
print GNUPLOT_DATA ($src_average_paths_180,"\t");

close(GNUPLOT_DATA);


#write header file
#print "generate gnu  plot...\n ";

open(GNUPLOT_HEADER,">$folder_temp/$header_file") ||
die "cant write gnuplot_header.tmp file";

print GNUPLOT_HEADER ("

#General Settings

set output \"$folder_temp/$output_file\"
set term postscript eps enhanced color
set size 0.7,0.7

#set xrange [0:1000]
#set yrange [0:20]
```

```
    # presentation
    set xrange [300:350]
    set yrange [0:2.5]

    set title \" Number of paths on average over time\" font \"Helvetica-Bold,14\"
    set xlabel \"time [s]\"
    set ylabel \"number of paths\"

# Presentation
#   plot    \"$folder_temp/$data_file_s_t\" using 1:2 ".
                "title \"# paths routing table\" with lines linetype 1 linewidth 2, \\
#           \"$folder_temp/$data_file_src\" using 1:2 ".
                "title \"# paths source \" with lines linetype 3 linewidth 2, \\
#           \"$folder_temp/$data_file_src\" using 1:3 ".
                "title \"# paths avg \" with lines linetype 4 linewidth 2

    plot    \"$folder_temp/$data_file_src\" using 1:2 ".
                "title \"# paths\" with lines linetype 1 linewidth 2, \\
            \"$folder_temp/$data_file_src\" using 1:3 ".
                "title \"# paths time avg \" with lines linetype 3 linewidth 2
    ");

    close(GNUPLOT_HEADER);

    # make graphic
    print `gnuplot $folder_temp/$header_file`;

    # view plot
    print `gv $folder_temp/$output_file`;

}

sub plot_path_shortest
{
#   plot number of paths found on average over shortest path

    my $INFINITY                    = 99999999;

    my $path_shortest_max           = $path_shortest{"shortest path max"};

    my $path_number_180_time_avg    = 0;
    my $path_number_180_time_std    = 0;

    my $path_number_180_max_avg     = 0;
    my $path_number_180_max_std     = 0;


    my $path_number_180_rp_p_avg    = 0;
    my $path_number_180_rp_p_std    = 0;

    my $connections                 = 0;
    my $route_requests              = 0;

    # gnuplot
    my $data_file                   = "gnuplot_data_stat_path_shortest.tmp";
    my $header_file                 = "gnuplot_header_stat_path_shortest.tmp";
    my $output_file                 = "stat_path_shortest_path.eps";

    #write data file
    print "generate gnu data file ... \n";

    open(GNUPLOT_DATA,">$folder_temp/$data_file") ||
    die "can't write gnuplot_data.tmp file";

    for(my $ii = 1; $ii <= $path_shortest_max; $ii++)
    {
        #update data
```

```
        $path_number_180_time_avg   = $path_shortest{$ii}{"180 time avg"};
        $path_number_180_time_std   = sqrt($path_shortest{$ii}{"180 time var"});

        $path_number_180_max_avg    = $path_shortest{$ii}{"180 max avg"};
        $path_number_180_max_std    = sqrt($path_shortest{$ii}{"180 max var"});

        $path_number_180_rp_p_avg   = $path_shortest{$ii}{"180 rp_p avg"};
        $path_number_180_rp_p_std   = sqrt($path_shortest{$ii}{"180 rp_p var"});

        $connections                = $path_shortest{$ii}{"num con"};
        $route_requests             = $path_shortest{$ii}{"180 rp_p num rreq"}/100;

        # plot new data at new time: (datapoint 2)

        # 01: shortest path
        print GNUPLOT_DATA ("$ii\t");

        # 180

        # 02: number of samples
        print GNUPLOT_DATA ("$connections\t");

        # 03: number of path found on average
        print GNUPLOT_DATA ("$path_number_180_time_avg\t");

        # 04: sigma = sqrt(variance) of 03
        printf GNUPLOT_DATA ("$path_number_180_time_std\t");

        # 05: max found path (avg over all connections)
        print GNUPLOT_DATA ("$path_number_180_max_avg\t");

        # 06: sigma = sqrt(variance) of 05
        print GNUPLOT_DATA ("$path_number_180_max_std\t");

        # 07 number of route request
        print GNUPLOT_DATA ("$route_requests\t");

        # 08: number of path found by one rreq
        print GNUPLOT_DATA ("$path_number_180_rp_p_avg\t");

        # 09: sigma = sqrt(variance) of 08
        printf GNUPLOT_DATA ("$path_number_180_rp_p_std\t");

        print GNUPLOT_DATA ("\n");
    }

    close(GNUPLOT_DATA);

    #write header file
    print "generate gnu header file...\n ";

    open(GNUPLOT_HEADER,">$folder_temp/$header_file") ||
    die "can't write gnuplot_header.tmp file";

    print GNUPLOT_HEADER ("
    #General Settings

    set output \"$folder_temp/$output_file\"
    set term postscript eps enhanced color
    set size 0.7,0.7

    set xrange [1:$path_shortest_max]
    set yrange [0:7]
    set y2range [0:20]

    set xtics 1, 1
    set ytics 0, 1
```

```
    set y2tics 0, 5

    set ytics nomirror

    set title \" Paths found on average for a shortest path with x hops (t>180) \"".
                  " font \"Helvetica-Bold,14\"
    set xlabel \"hops\"
    set ylabel \"number of paths\"
    # Presentation
    # set y2label \"number of samples\"
    set y2label \"number of connections\"

    # Presentation
    #plot   \"$folder_temp/$data_file\"  using 1:5:6 axis x1y1 ".
               "title \"paths max\"with yerrorlines linetype 1 linewidth 2,  \\
    #       \"$folder_temp/$data_file\"  using 1:3:4 axis x1y1 ".
               "title \"paths avg\"with yerrorlines linetype 3 linewidth 2,  \\
    #       \"$folder_temp/$data_file\"  using 1:8:9 axis x1y1 ".
               "title \"paths after one RREQ\"with yerrorlines linetype 4 linewidth 2,\\
    #       \"$folder_temp/$data_file\"  using 1:2   axis x1y2 ".
               "title \" #connections \"with lines linetype 5 linewidth 2,\\
    #       \"$folder_temp/$data_file\"  using 1:7   axis x1y2 ".
               "title \" #rreq (in 100)\" with lines linetype 8 linewidth 2 \\

    plot    \"$folder_temp/$data_file\"  using 1:3:4 axis x1y1 ".
               "title \"paths avg on time\"with yerrorlines linetype 3 linewidth 2,  \\
            \"$folder_temp/$data_file\"  using 1:8:9 axis x1y1 ".
               "title \"paths after one RREQ\"with yerrorlines linetype 4 linewidth 2,\\
            \"$folder_temp/$data_file\"  using 1:2   axis x1y2 ".
               "title \" # connections \"with lines linetype 5 linewidth 2\\
    ");

    close(GNUPLOT_HEADER);

    # make graphic
    print `gnuplot $folder_temp/$header_file`;

    # view plot
    print `gv $folder_temp/$output_file`;
}

sub dropzone
{
    # plot dropped packet

    # gnuplot
    my $data_file       = "gnuplot_data_dropzone.tmp";
    my $header_file     = "gnuplot_header_dropzone.tmp";
    my $output_file     = "dropzone.eps";

    #write data file
    print "generate gnu data file ... \n";

    open(GNUPLOT_DATA,">$folder_temp/$data_file") ||
        die "can't write gnuplot_data.tmp file";

    for(my $ii = 0; $ii <$node_total; $ii++)
    {
        # 01: node number #a
        print GNUPLOT_DATA ($ii,"\t");

        # 02: X position
        print GNUPLOT_DATA ($position_x_node[$ii],"\t");

        # 03: Y position
        print GNUPLOT_DATA ($position_y_node[$ii],"\t");
```

```
        # 04: dropped data packet of node #a over total sent data packet
        print GNUPLOT_DATA ($dropped_dpackets_v_tot_sent[$ii],"\t");

        # 05: dropped data packet of node #a over received data packet by node #a
        print GNUPLOT_DATA ($dropped_dpackets_v_received_node[$ii],"\t");

        print GNUPLOT_DATA ("\n");
    }
    close(GNUPLOT_DATA);

    #write header file
    print "generate gnu header ...\n ";

    open(GNUPLOT_HEADER,">$folder_temp/$header_file") ||
        die "can't write gnuplot_header.tmp file";

    print GNUPLOT_HEADER ("

    #General Settings

    set output \"$folder_temp/$output_file\"
    set term postscript eps enhanced color

    set size 1,2
    set origin 0,0

    set multiplot


# Multiplot MAP
#
#

    set nokey
    set pm3d
    set size square

    set view map

    #make a grid out of the scatter-data
    #(be carefull with paramater settings)
    # <row_size> <col_size> <norm>
    # (like a low pass filter)

    set dgrid3d 100, 100, 16

    set xrange [0:1600]
    set yrange [0:1600]
    set zrange [0:1]

    set size 1,1

    set xlabel \"X\"
    set ylabel \"Y\"


    set xtics 500
    set ytics 500


# lost data packet over all sent

    set title \"Dropped data packets over total sent data packets\" ".
            "font \"Helvetica-Bold,14\"

    set origin 0,1
    set cbrange [0:0.01]
```

```
    splot \"$folder_temp/$data_file\" using 2:3:4 with pm3d


# lost data packet over received

    set title \"Dropped data packet over received data packet of this node\" ".
            " font \"Helvetica-Bold,14\"

    set origin 0,0

    set cbrange [0:0.10]

    splot \"$folder_temp/$data_file\" using 2:3:5 with pm3d

#end MAP

    unset dgrid3d
    unset pm3d

# node position marker

    set title \" \"
    set xlabel \" \"
    set ylabel \" \"
    set noxtic
    set noytic
    set nokey
    set pointsize 1

#map 1
    set origin 0,1
    splot \"$folder_temp/$data_file\" using 2:3:4 with points  7 7

#map 2
    set origin 0,0

# INFOS:

    splot \"$folder_temp/$data_file\" using 2:3:5 with points  7 7

    set xtic
    set ytic
    set key

set nomultiplot

    \n");


    close(GNUPLOT_HEADER);

    # make graphic
    print `gnuplot $folder_temp/$header_file`;

    # view plot
    print `gv $folder_temp/$output_file`;
}

sub plot_path_space
{
    # plot paths in rtable space time for $connection for $time_slot

    # interface

    my $connection  = shift(@_);
    my $plot_name   = shift(@_);
    my @active_path = @{shift(@_)};
```

```
my $label       = shift(@_);

my $src         = $src_dest{$connection}{"src"};
my $dst         = $src_dest{$connection}{"dst"};

my $path_total  = $src_dest_path{$src}{$dst}{"num paths"};

# gnuplot
my $data_file       = "gnuplot_data_paths_rtable";
my $data_file_back  = "gnuplot_data_paths_rtable_back"; # background
my $header_file     = "gnuplot_header_paths_rtable.tmp";
my $output_file     = "$plot_name";

# variables

my @hops_array = (); # save hops of path

my $src_x       = $position_x_node[$src];
my $src_y       = $position_y_node[$src];
my $dst_x       = $position_x_node[$dst];
my $dst_y       = $position_y_node[$dst];

#DEBUG OUT PARAMETER CHECK
if($DEBUG == 1)
{
    printf("\n-- plot_path_space --\n");
    printf("SRC %4.i (%6.3f,%6.3f) DEST %4.2i (%6.3f,%6.3f) \n",
        $src, $src_x,$src_y, $dst, $dst_x,$dst_y);
    printf("Filename: %s \n",$output_file);
}

#write data file (3D)

# print "generate gnu data file ... \n";
# LINKS

open(GNUPLOT_DATA,">$folder_temp/$data_file") || die "can't write $data_file file";

# loop over paths of @active_path
for(my $loop_path = 1; $loop_path <= $path_total; $loop_path++)
{# over different paths
    if($active_path[$loop_path] > 0) # path active?
    {#yes
        @hops_array = split(/-/,$src_dest_path{$src}{$dst}{$loop_path});

        for(my $loop_hop = 0; $loop_hop < scalar(@hops_array); $loop_hop++)
        { # write data
            if($hops_array[$loop_hop] =~"D" && $hops_array[$loop_hop] =~"S")
            { # Header

            }
            elsif($hops_array[$loop_hop] =~ m/(0|1|2|3|4|5|6|7|8|9)/)
            { # write line from last hop to this hop
                print GNUPLOT_DATA ("$position_x_node[$hops_array[$loop_hop]] \t".
                        " $position_y_node[$hops_array[$loop_hop]] \t 1 \n");
            }
        }
        print GNUPLOT_DATA ("\n\n");
    }
}

close(GNUPLOT_DATA);

# BACKGROUND

open(GNUPLOT_DATA,">$folder_temp/$data_file_back") ||
die "can't write $data_file_back file";
```

```
for(my $ii = 0; $ii <$node_total; $ii++)
{
    # 01: node number #a
    print GNUPLOT_DATA ($ii,"\t");

    # 02: X position
    print GNUPLOT_DATA ($position_x_node[$ii],"\t");

    # 03: Y position
    print GNUPLOT_DATA ($position_y_node[$ii],"\t");

    #04: dropped data packet of node #a over total sent data packet
    print GNUPLOT_DATA ($dropped_dpackets_v_tot_sent[$ii],"\t");

    # 05: dropped data packet of node #a over received data packet by node #a
    print GNUPLOT_DATA ($dropped_dpackets_v_received_node[$ii],"\t");

    print GNUPLOT_DATA ("\n");
}
close(GNUPLOT_DATA);

#write header file
#print "generate gnu  plot...\n ";

open(GNUPLOT_HEADER,">$folder_temp/$header_file") ||
die "can't write gnu header file $header_file";

print GNUPLOT_HEADER ("
#General Settings

set output \"$output_file\"
set term postscript eps enhanced color

set size 1,1
set origin 0,0

set multiplot

set nokey
set size square
set view map

set xrange [0:1700]
set yrange [0:1600]
set zrange [0:10]

set xlabel \"X\"
set ylabel \"Y\"

set xtics 500
set ytics 500

set title \"Content of routing tables\" font \"Helvetica-Bold,14\"

# BACKGROUND

## dropped packet
set pm3d
set size square
set dgrid3d 100,100,16
set cbrange [0:0.025]

splot \"$folder_temp/$data_file_back\" using 2:3:4 with pm3d

unset dgrid3d
unset pm3d
```

```
    ## label

    set label 1 \"$label    \"at 0, -300 left front

    ## source

    set arrow 1 from 0, 0, 20 to $src_x, $src_y, 20 back nofilled linetype 1 linewidth 2

    ## destination

    set arrow 2 from 0, 0, 20 to $dst_x, $dst_y, 20 back nofilled linetype 3 linewidth 2

    ## all nodes

    set noxtic
    set noytic
    set nokey
    set pointsize 1
    splot \"$folder_temp/$data_file_back\" using 2:3:4 with points  7 7

    set noxtic
    set noytic
    set nokey
    set pointsize 1
    splot \"$folder_temp/$data_file_back\" using 2:3:4 with points  3 3

    # LINKS

    set pointsize 2

    splot \"$folder_temp/$data_file\" using 1:2:3 with lines  1 6

    # node position marker

    set title \" \"
    set xlabel \" \"
    set ylabel \" \"
    set noxtic
    set noytic
    set nokey
    set pointsize 2

    splot \"$folder_temp/$data_file\" using 1:2:3 with points  1 6

    \n");

    close(GNUPLOT_HEADER);

    # make graphic
    print `gnuplot $folder_temp/$header_file`;
}
```

Listing A.11: analysis_one_run.pl

A.3.6. analysis_multiple_runs.pl

```
#!/usr/bin/perl
#-------------------------------------------------------------------------------
# ANALYSIS_MULTIPLE_RUNS.PL       FILE TO GENERATE STATISTICS OVER DIFFERENT .SIM FILES
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [Perl Script]
#
# Date:     July 2006
#
# Input: ['simulation data result folder']
#
#
# Output: graphics in eps format
#
#
# Idea:
#       This script calculates statistic data out of different
#       'simulation results files' (.sim) and plot this values with
#       the help of gnuplot
#
# Concept:
#       A plot can be characterize by the three dimension x-axis, y-axis and
#       the plotted functions (z-dim)
#
#     Y-Dim ^    - Z-Dim
#       |   -
#       |
#       |_________> X-Dim
#
#       We added an additionally filter, called Fix-Filter, to select the Version
#       of the simulator or different topologies.
#
#       To generate a plot with this script this four filters have to be specified.
#       The script then searches in the folder of the simulation results for
#       the needed files and calculates all data points.
#
#
# used command line tools:
#
#   Filter:     Filter function based on 'ls' and 'grep'
#
#   Plot:       Plots are generated with 'gnuplot'
#
# Implementation:
#
#   sub set_filter      This function inits the filters
#
#   sub file_filter     This function selects the needed files to build the statistics
#
#   sub statistic       This function calculates the statistics over different sim-files
#
#   sub gnuplot         This function plots the statistics
#
# Usage:
#   Set the parameter in the SET PARAMETER part
#   If needed change the filter in the set_filter function
#
# Known Bugs:
#   No Bugs reported (20.07.2006)
#-------------------------------------------------------------------------------
```

```
# behave like a reasonable programming language
use strict;

# Declaration of global variables
#-------------------------------------------------

# Filter
my $choice_fix;      # select fix filter
my $choice_z;        # select z filter
my $choice_x;        # select x filter
my $choice_y;        # select y filter

# FIX
my $fix_grep;        # grep pattern for the fix filter
my $fix_title;       # first part of the title
my $fix_file_name;   # part of the output file name

# Z-Value
my @z_dim_grep;      # grep pattern for the z filter
my @z_label;         # label for the z-dimension
my $z_file_name;     # part of the output file name

# X-Value
my @x_dim_grep;      # grep pattern for the x filter
my @x_data_point;    # datapoint to pattern
my $x_title;
my $x_axis;          # label axis in gnuplot
my $x_tics;          # tics in gnuplot
my $x_range;         # range x axis
my $x_file_name;     # part of the output file name

# Y-Value
my $y_title;
my $y_axis;          # label axis in gnuplot
my $y_range;         # range y axis
my $y_tics;          # tics in gnulot
my $y_file_name;     # part of the output file name

# Statistic container
my $stat_type;       # chose type of statistic
my @statistic_container;# statistic container
    # 2D statistic [z-dim][x-dim][...data...]

# File container
my @file_to_stat;    # array with filenames that


# ---------- SET PARAMETER ---------- START

# switch DEBUG on/off
my $DEBUG = 0;

# Version & Simulation data
my $version = "D001";
my $simulation_results = " ../sim_data/$version";
my $simulation_result_type = "sim";

# Check if simulation run until the end
# (Is sim file OK?)

my $t_first_data = 180;  # time when all data connection should be up
my $t_last_data = 1000;  # time when the simulation is stopped

# temporal working folder...
my $folder_temp = "temp";
mkdir $folder_temp;
```

```
# select filter

$choice_fix = 'random movement';
#$choice_fix     = 'hexa hotspot';
#$choice_fix     = 'hexa homogen';
#$choice_fix     = 'circle random';

$choice_z    = 'only4';
#$choice_z   = 'all';

$choice_x    = 'load';
#$choice_x   = 'load_fine';
#$choice_x   = 'node density';
#$choice_x   = 'movement pause time';
#$choice_x   = 'movement speed';

$choice_y    = 'ratio';
#$choice_y   = 'delay';

# ---------- SET PARAMETER ---------- END


# ---------- MAIN ----------

#set filter
set_filter();

#select files and calculate statistic for this point
file_filter();

# plot figure

# overwrite range if needed
$y_range     ="0.2:1";
$x_range     ="1:4";

gnuplot();


# ---------- sub function ----------
sub set_filter
{
    printf("set filter ...\n");

    # set the filter to select the correct .sim files
    # filter works on the name of the .sim files
    #-----------------------------------------------

    # file name:
    #-----------
    # [description_short]-
    # -ri-[number_of_run]
    # -qt-[queue_type]
    # -ql-[queue_length]
    # -dx-[dimension_x]
    # -dy-[dimension_y]
    # -nn-[number_node]
    # -sd-[sim_duration]
    # -sp-[sim_progress]
    # -pt-[pause_time]
    # -ms-[maximum_speed]
    # -scp-[scen_type]
    # -st-[source_type]
    # -mc-[max_connection]
    # -ps-[pkt_size]
```

```
# -pr-[pkt_rate]
# -sc-[sched_type]
# -pn-[aomdv_max_path]
# .sim  [simulation result file type]
#
# example:
# V010-ri-9-qt-0-ql-50-dx-2000-dy-2000-nn-80-sd-1000-sp-100-pt-0-ms-0.0-  ...
#  ... scp-40-st-cbr-mc-50-ps-256-pr-3.0-sc-1-pn-10.sim


# set fix parameters
#-------------------
# Version eg V002       ="V001.*"
# number_of_run         ="ri -.*";
# queue_type            ="qt-0-.*";
# queue_length          ="ql-50-.*";
# dimension_x           ="dx-1500-.*";
# dimension_y           ="dy-1500-.*";
# simulation_duration   ="sd-1000-.*";
# sim_progress          ="sp-10-.*";
# source_type           ="st.*cbr.*";
# ending                ="sim";

my $queue_type          = 0;
my $queue_length        = 50;
my $dimension_x         = 2000;
my $dimension_y         = 2000;
my $sim_duration        = 1000;
my $sim_progress        = 100;
my $source_type         = 'cbr';


if($choice_fix eq "hexa homogen")
{

    # scenario 30

    $dimension_x            = 2000;
    $dimension_y            = 2000;

    $fix_grep = "$version".
                ".*ri".
                "-.*qt-$queue_type".
                "-.*ql-$queue_length".
                "-.*dx-$dimension_x".
                "-.*dy-$dimension_y".
                "-.*sd-$sim_duration".
                "-.*sp-$sim_progress".
                "-.*scp-30".
                "-.*st.*$source_type".
                ".*$simulation_result_type";
    $fix_title = "hexagon homogen ($dimension_x m x $dimension_y m)";
    $fix_file_name = "[$version-dx-$dimension_x-dy-$dimension_y-hexa-homogen]";
}
elsif($choice_fix eq "hexa hotspot")
{   #scenario 40

    $dimension_x            = 2000;
    $dimension_y            = 2000;


    $fix_grep = "$version".
                ".*ri".
                "-.*qt-$queue_type".
                "-.*ql-$queue_length".
                "-.*dx-$dimension_x".
                "-.*dy-$dimension_y".
```

```
                "-.*sd-$sim_duration".
                "-.*sp-$sim_progress".
                "-.*scp-40".
                "-.*st.*$source_type".
                ".*$simulation_result_type";
    $fix_title = "hexagon hotspot $dimension_x m x $dimension_y m";
    $fix_file_name = "[$version-dx-$dimension_x-dy-$dimension_y-hexa-hotspot]";
}
elsif($choice_fix eq "circle random")
{
    #scenario 50

    $dimension_x            = 2000;
    $dimension_y            = 2000;

    $fix_grep = "$version".
                ".*ri".
                "-.*qt-$queue_type".
                "-.*ql-$queue_length".
                "-.*dx-$dimension_x".
                "-.*dy-$dimension_y".
                "-.*sd-$sim_duration".
                "-.*sp-$sim_progress".
                "-.*scp-50".
                "-.*st".
                ".*$source_type".
                ".*$simulation_result_type";
    $fix_title = "circle random $dimension_x m x $dimension_y m";
    $fix_file_name = "[$version-dx-$dimension_x-dy-$dimension_y-circle-random]";
}
elsif($choice_fix eq "random movement")
{
    #scenario 00
    #
    $dimension_x            = 1000;
    $dimension_y            = 1000;

    $fix_grep = "$version".
                ".*ri".
                "-.*qt-$queue_type".
                "-.*ql-$queue_length".
                "-.*dx-$dimension_x".
                "-.*dy-$dimension_y".
                "-.*sd-$sim_duration".
                "-.*sp-$sim_progress".
                "-.*scp-00".
                "-.*st".
                ".*$source_type".
                ".*$simulation_result_type";
    $fix_title = "random movement $dimension_x m x $dimension_y m";
    $fix_file_name = "[$version-dx-$dimension_x-dy-$dimension_y-circle-random]";
}

else
{
    die "set_grep_parameter: Can't resolve \$choice_fix";
}

# set Z parameter
# ---------------
# sched_type        ="sc-.*";
# aomdv_max_path    ="pn-.*";

if($choice_z eq "all")
{
    # grep pattern for the z filter
    @z_dim_grep = ( "sc-0-.*pn-1[[:punct:]]$simulation_result_type",
```

```
                    "sc-0-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-1-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-2-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-3-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-4-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-5-.*pn-10[[:punct:]]$simulation_result_type");

    # label for the z-dimension
    @z_label = ("AODV  1P",
                "AOMDV  10P",
                "RR  10P",
                "WRR  10P",
                "Sel  10P",
                "RTT 10P",
                "NB 10P");

    # part of the filename
    $z_file_name =  "[aodv-aomdv-rr-wrr-sel-rtt-neigh-mp-10]";
}
elsif($choice_z eq "only4")
{
    # grep pattern for the z filter
    @z_dim_grep = ( "sc-0-.*pn-1[[:punct:]]$simulation_result_type",
                    "sc-0-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-1-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-2-.*pn-10[[:punct:]]$simulation_result_type",
                    "sc-3-.*pn-10[[:punct:]]$simulation_result_type");

    # label for the z-dimension
    @z_label = ("AODV",
                "AOMDV",
                "RR",
                "WRR",
                "Sel");

    # part of the filename
    $z_file_name = "[aodv-aomdv-rr-wrr-sel]";
}

elsif($choice_z eq "manual")
{ # DEBUG FILTER
    @z_dim_grep = ("sc-3-.*pn-15");
    @z_label    = ("manual");
    $z_file_name = "[manual]";
}
else
{
    die "set_grep_parameter: Can't resolve \$choice_y";

}

# set X parameter
#-----------------
# number_node       ="nn-.*";
# pause_time        ="pt-10-.*";
# maximum_speed     ="ms-10[[:punct:]]0-.*";
# connection        ="mc-20-.*";
# pkt_size          ="ps-256-.*";
# pkt_rate          ="pr-1[[:punct:]]0-.*";

# use this variables to easily adapt filter
my $nn = 50;    # number of nodes
my $pt = 0;     # pause time
my $ms_H = 5;   # speed (head)
my $ms_T = 0;   # speed (tail)
my $mc = 50;    # maximum number of connections
my $ps = 256;   # pkt size
```

```
my $pr_H = 1;   # pkt rate (head)
my $pr_T =0;    # pkt rate (tail)

#use "\[" => due to $a[[ is interpretation as array...


if ($choice_x eq "node density")
{
    @x_dim_grep =(
    "nn-30-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*".
            "pr-$pr_H\[\[:punct:]]$pr_T-",

    "nn-50-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*".
            "pr-$pr_H\[\[:punct:]]$pr_T-",

    "nn-70-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*".
            "pr-$pr_H\[\[:punct:]]$pr_T-",

    "nn-100-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*".
            "pr-$pr_H\[\[:punct:]]$pr_T-");

    @x_data_point = (30,50,70,90);

    # axis in gnuplot
    $x_axis     = "numbers of nodes ".
        "(pt = $pt, ms = $ms_H.$ms_T, mc = $mc, pr = $pr_H.$pr_T, ps = $ps)";
    $x_tics     = "30,50,70,90";
    $x_range    = "0:90";
    $x_title    = "node density";
    $x_file_name= "[nodes--nn-XX-pt-$pt-ms-$ms_H.$ms_T-mc-$mc-ps-$ps-pr-$pr_H.$pr_T]";
}
elsif ($choice_x eq "movement pause time")
{
    @x_dim_grep =(
    "nn-$nn-.*pt-0-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*".
            "pr-$pr_H\[\[:punct:]]$pr_T-",

    "nn-$nn-.*pt-30-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*pr-".
            "$pr_H\[\[:punct:]]$pr_T-",

    "nn-$nn-.*pt-120-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*pr-".
            "$pr_H\[\[:punct:]]$pr_T-",

    "nn-$nn-.*pt-300-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*pr-".
            "$pr_H\[\[:punct:]]$pr_T-",

    "nn-$nn-.*pt-600-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*pr-".
            "$pr_H\[\[:punct:]]$pr_T-",

    "nn-$nn-.*pt-900-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*ps-$ps-.*pr-".
            "$pr_H\[\[:punct:]]$pr_T-");

    @x_data_point = (0,30,60,120,600,900);

    # axis in gnuplot
    $x_axis     = "pause time [s] - max speed $ms_H.$ms_T m/s ".
                  "(nn = $nn, mc = $mc, ps = $ps, pr = $pr_H.$pr_T)";
    $x_tics     = "0,30,60,120,600,900";
    $x_range    = "0:1000";
    $x_title    = "movement";
    $x_file_name=
    "[movement--nn-$nn-pt-XXX-ms-$ms_H.$ms_T-mc-$mc-ps-$ps-pr-$pr_H.$pr_T]";
}
elsif ($choice_x eq "movement speed")
{
    @x_dim_grep =(
    "nn-$nn-.*pt-$pt-.*ms-1[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
```

```
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-2[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-3[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-4[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-5[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-6[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-7[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-8[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-9[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-",

        "nn-$nn-.*pt-$pt-.*ms-9[[:punct:]]0-.*mc-$mc-.*ps-$ps-.*".
                "pr-$pr_H\[\[:punct:]]$pr_T-");

        @x_data_point = (1,2,3,4,5,6,7,8,9,10);

        # axis in gnuplot
        $x_axis     = "max speed [m/s] ".
            " (pt = $pt, nn = $nn, mc = $mc, ps = $ps, pr = $pr_H.$pr_T)";
        $x_tics     = "0,2,4,6,8,10";
        $x_range    = "0:10";
        $x_title    = "movement";
        $x_file_name=
        "[movement--nn-$nn-pt-XXX-ms-$ms_H.$ms_T-mc-$mc-ps-$ps-pr-$pr_H.$pr_T]";
    }

    elsif ($choice_x eq "load")
    {
        @x_dim_grep =   (
        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-1[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-1[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-2[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-2[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-3[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-3[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-4[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-4[[:punct:]]5-",
```

```
        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-5[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-5[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-6[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-6[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-7[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-8[[:punct:]]0-"
                        );
        @x_data_point = (1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,8);

        # axis in gnuplot
        $x_axis      = "packet rate [packets/s]  ".
                "(max con = $mc; ps = $ps; nn = $nn; pt = $pt; ms = $ms_H.$pt)";
        $x_tics      = "0,1,2,3,4,5,6,7,8";
        $x_range     = "0:10";
        $x_title     = "load";
        $x_file_name= "[load--nn-$nn-pt-$pt-ms-$ms_H.$ms_T-mc-$mc-ps-$ps-pr-X.0]";
}
elsif ($choice_x eq "load_fine")
{
        @x_dim_grep =   (
        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-1[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-1[[:punct:]]25-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-1[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-1[[:punct:]]75-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-2[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-2[[:punct:]]25-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-2[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-2[[:punct:]]75-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-3[[:punct:]]0-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-3[[:punct:]]25-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-3[[:punct:]]5-",

        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-3[[:punct:]]75-",
```

```
        "nn-$nn-.*pt-$pt-.*ms-$ms_H\[\[:punct:]]$ms_T-.*mc-$mc-.*".
                "ps-$ps-.*pr-4[[:punct:]]0-"

                                    );
        @x_data_point = (1,1.25,1.5,1.75,2,2.25,2.5,2.75,3.0,3.25,3.5,3.75,4.0);

        # axis in gnuplot
        $x_axis     = "packet rate [packets/s]  ".
                "(max con = $mc; ps = $ps; nn = $nn; pt = $pt; ms = $ms_H.$pt)";
        $x_tics     = "0,1,2,3,4";
        $x_range    = "0:4";
        $x_title    = "load";
        $x_file_name= "[load--nn-$nn-pt-$pt-ms-$ms_H.$ms_T-mc-$mc-ps-$ps-pr-X.0]";
    }

    elsif($choice_x eq "manual")
    {
        die " manual not set ...";
    }
    else
    {
        die "set_grep_parameter: Can't resolve \$choice_x";
    }


    # set Y parameter
    #-----------------
    if ($choice_y eq "ratio")
    {
        $y_axis     = "delivery ratio [data pkt received / data pkt sent ]";
        $y_tics     = "0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1";
        $y_range    = "0:1";
        $y_title    = "Delivery ratio";
        $y_file_name= "ratio";
        $stat_type  = "delivery_ratio_mean";
    }
    elsif ($choice_y eq "delay")
    {
        $y_axis     = "end to end delay [s]";
        $y_tics     = "0";
        $y_range    = "0:0.5";
        $y_title    = "End to end delay";
        $y_file_name= "delay";
        $stat_type  = "pkt_delay_mean";
    }
    else
    {
        die "set_grep_parameter: Can't resolve \$choice_y";
    }


    # DEBUG OUTPUT:
    if($DEBUG==1)
    {
        printf("FIX:\t %s\nZ:\t%s\nX:\t%s\nZ:\t%s\n",
        $fix_grep,
        $z_dim_grep[0],
        $x_dim_grep[0],
        $stat_type);
    }
}


sub file_filter
{
    # This function search for all datapoints the corresponding sim files and
```

```
# call and save the statistic value for this point in the global statistic container

# 1. list all files of the simulation result folder
# 2. perform Fix-Filter
# 3. perform Z-Filter
# 4. perform X-Filter
# 5. perform Y-Filter
# 6. save the selected sim files in a global file container
# 7. call 'statistic()' --> calculates the statistic for
# this point (use the global file container)
# 8. save this value in the global statistic container
# 9. repeat this for all data points



# loop variables
my $z_dim_index = 0;     #loop variable in z dim
my $x_dim_index = 0;     #loop variable in x dim
my @y;                   #loop statistic variable

print "calculate statistic ...\n";

# get all possible .P files and save list in $folder_temp/ls_all

print `rm -f $folder_temp/ls_all`;
print `ls $simulation_results > $folder_temp/ls_all`;

# FIX Filter
print `rm -f $folder_temp/list_fix`;     #remove existing file

print ("FIX Filter: ",`grep -c $fix_grep $folder_temp/ls_all`,"\n");
print `grep $fix_grep $folder_temp/ls_all > $folder_temp/list_fix`;


# Z-Filter

for($z_dim_index = 0; $z_dim_index < scalar(@z_dim_grep) ; $z_dim_index++)
{
    print `rm -f $folder_temp/list_z`; #remove existing file

print ("\t Z Filter: ", `grep -c '$z_dim_grep[$z_dim_index]' $folder_temp/list_fix`,
       "\t\t\t\t ( $z_dim_grep[$z_dim_index])\n");

print `grep '$z_dim_grep[$z_dim_index]' $folder_temp/list_fix > $folder_temp/list_z`;

    for($x_dim_index= 0; $x_dim_index<scalar(@x_dim_grep); $x_dim_index++)
    {
        # X-Filter

        @file_to_stat = ();

        print ("\t\t X Filter: ",
                `grep -c '$x_dim_grep[$x_dim_index]' $folder_temp/list_z` ,
                " \t\t\t\t ($x_dim_grep[$x_dim_index])\n");

        @file_to_stat = `grep '$x_dim_grep[$x_dim_index]' $folder_temp/list_z `;


        # Calculate Statistic over selected files
        if (scalar(@file_to_stat)== 0)
        {
            print "\t\t\t no files selected, set value to -1 \n";
            $statistic_container[$z_dim_index][$x_dim_index]= [(-1,0)];
        }
        else
        {
            # calculate statistic of the selected .P files
```

```
                $statistic_container[$z_dim_index][$x_dim_index] = [statistic()];
            }
        }
    }
}

# ———————— calc stat over given file_to_stat_array ————————

sub statistic{

    # read the simulation result files and calculate the statistic

#————————————————————————————————
    # File parser, handler
    my $file_index = 0;
    my @lines;
    my $line;
    my @elements;

    # number of nodes
    my $number_of_nodes_total = 0;
    my $number_of_nodes_avg = 0;        # avg over run

    # data delivery ratio (received/transmitted)
    my $data_delivery_ratio_loop = 0;
    my $data_delivery_ratio_avg = 0;    # avg over different runs
    my $data_delivery_ratio_var = 0;    # var over different runs

    # end to end pkt delay
    my $end_to_end_pkt_delay_loop = 0;
    my $end_to_end_pkt_delay_avg = 0; # avg over different runs
    my $end_to_end_pkt_delay_var = 0; # var over different runs

    # Check if simulation result file is compled
    # (CORE BUG CHECK)
    my $number_of_good_files = 0;

    my $data_pkt_first_time = 0;
    my $data_pkt_last_time = 0;
    my $last_time_stamp = 0;


    #return y value (statistic)
    my @y; #[1]=mean [2]=var


#————————————————————————————————

    # set loop variables to zero
    $data_delivery_ratio_loop = 0;
    $end_to_end_pkt_delay_loop = 0;
    $number_of_good_files = 0;

    for ($file_index = 0; $file_index < scalar(@file_to_stat); $file_index++)
    {
        $data_pkt_first_time = 0;
        $data_pkt_last_time = 0;
        $last_time_stamp = 0;

        open(SIM_FILE, "< $simulation_results/$file_to_stat[$file_index]") # open file
        or die "Couldn't open  $file_to_stat[$file_index] for reading: $!\n";

        (@lines) = <SIM_FILE>;

        foreach $line (@lines)
        {
            #split line in elements
```

```
        @elements = split(/ /, $line);

        if (($elements[0] =~ m/(0|1|2|3|4|5|6|7|8|9)/) && (scalar(@elements) > 3))
        {
            # number of nodes
            $number_of_nodes_total++;
        }

        if ($elements[0] =~ /RATIO/)
        {
            # delivered data
            $data_delivery_ratio_loop += $elements[1];

        }
        if ($elements[0] =~ /AVRG TOTAL delay of a data packet/)
        {
            # end to end pkt delay
            $end_to_end_pkt_delay_loop += $elements[1];
        }
        if ($elements[0] =~ /first data with id/ && scalar(@elements) < 8)
        {
            #first data pkt sent
            $data_pkt_first_time = $elements[3];

        }
        if ($elements[0] =~ /last data with id/ && scalar(@elements) < 8)
        {
            #last data pkt sent
            $data_pkt_last_time= $elements[3];
        }
    }
    close(SIM_FILE) or die "Couldn't close $file_to_stat[$file_index]\n";

    #check if simulation run until the end
    if(($data_pkt_first_time < $t_first_data + 10)
        && ($data_pkt_last_time > $t_last_data - 10 ))
    {
        $number_of_good_files++;
    }
    else
    {
        print "First data pkt sent at: $data_pkt_first_time\n";
        print "First data pkt sent at: $data_pkt_last_time\n";
        print "File Name: $file_to_stat[$file_index]\n";
    }
}


# check if all simulation files were OK
if($number_of_good_files != $file_index)
{
    die "Some .sim files are corrupt!!$!\n";
}


# calculate avgs:
$number_of_nodes_avg        = $number_of_nodes_total/scalar(@file_to_stat);
$data_delivery_ratio_avg    = $data_delivery_ratio_loop/scalar(@file_to_stat);
$end_to_end_pkt_delay_avg   = $end_to_end_pkt_delay_loop/scalar(@file_to_stat);


# collect data for var

# set loop variables to zero
$data_delivery_ratio_loop = 0;
$end_to_end_pkt_delay_loop = 0;
```

```
for ($file_index = 0; $file_index < scalar(@file_to_stat); $file_index++)
{
    open(SIM_FILE, "< $simulation_results/$file_to_stat[$file_index]") # open file
    or die "Couldn't open  $file_to_stat[$file_index] for reading: $!\n";

    (@lines) = <SIM_FILE>;

    foreach $line (@lines)
    {
        #split line in elements
        @elements = split(/ /, $line);

        if ($elements[0] =~ /RATIO/)
        {
            # delivered data variance
            $data_delivery_ratio_loop += ($elements[1]- $data_delivery_ratio_avg)**2;
        }

        if ($elements[0] =~ /AVRG TOTAL delay of a data pkt/)
        {
            # end to end pkt delay
        $end_to_end_pkt_delay_loop += ($elements[1] - $end_to_end_pkt_delay_avg)**2;
        }
    }
    close(SIM_FILE) or die "Couldn't close $file_to_stat[$file_index]\n";
}


# calculate var

$data_delivery_ratio_var  = $data_delivery_ratio_loop / scalar(@file_to_stat);
$end_to_end_pkt_delay_var = $end_to_end_pkt_delay_loop / scalar(@file_to_stat);

#DEBUG
if($DEBUG == 1)
{
    #files
    print("Number of 'runs': \t", scalar(@file_to_stat),"\n");
    print("Number of 'good' files: \t\t$number_of_good_files\n");
    print("Used files: \n @file_to_stat \n");
    #values
    print("Average number of nodes: \t\t$number_of_nodes_avg\n");
    print("Data delivery ratio in average:\t\t\t $data_delivery_ratio_avg\n");
    print("Data delivery ratio variance:\t\t $data_delivery_ratio_var\n");
    print("End to end pkt delay in average:\t\t $end_to_end_pkt_delay_avg\n");
    print("End to end pkt delay variance:\t\t $end_to_end_pkt_delay_var\n");
}

# select return value

if($stat_type eq "delivery_ratio_mean")
{
    # give back delivery ratio
    @y = ($data_delivery_ratio_avg, $data_delivery_ratio_var, scalar(@file_to_stat));

}
elsif($stat_type eq "pkt_delay_mean")
{
    # give back end to end pkt delay
@y = ($end_to_end_pkt_delay_avg, $end_to_end_pkt_delay_var, scalar(@file_to_stat));

}
else
{
    die "'sub:get_standard_mean': $stat_type is not implemented \n";
}
return(@y);
```

```
}

# ———————— Gnuplot ————————

sub gnuplot{
    # write a single plot

    my $gnu_data_file = "gnu_data_singleplot.tmp";
    my $gnu_header_file ="gnu_header_singleplot.tmp";
    my $gnu_output_file = "$fix_file_name-$z_file_name-$x_file_name-$y_file_name.eps";

    # write data file
    #————————————————

    print "generate gnu data file ... \n";

    open(GNUPLOT_DATA,">$folder_temp/$gnu_data_file")
        or die "can't write $folder_temp/$gnu_data_file file";

    for(my $i = 0; $i < scalar(@x_dim_grep); $i++)
    {
        print GNUPLOT_DATA "$x_data_point[$i] \t";

        for(my $j = 0; $j< scalar(@z_dim_grep); $j++)
        {
            printf GNUPLOT_DATA ("0%f \t", $statistic_container[$j][$i][0]);    # mean
            printf GNUPLOT_DATA ("0%f \t", sqrt($statistic_container[$j][$i][1]));  # stad
            printf GNUPLOT_DATA ("0%f \t", $statistic_container[$j][$i][2]);    # samples
        }
        print GNUPLOT_DATA "\n";
    }

    close(GNUPLOT_DATA) or die "can't close $folder_temp/$gnu_data_file file";

    # write header file
    #————————————————

    print "generate gnu header file ...\n";

    open(GNUPLOT_HEADER,">$folder_temp/$gnu_header_file")
        or die "can't write $folder_temp/$gnu_header_file";

    print GNUPLOT_HEADER "
    # output file type:
    set term postscript eps enhanced color

    # output file
    set output \"$folder_temp/$gnu_output_file\"

    # plot size
    set size 0.8,0.8

    # labels
    set title  \" $y_title versus $x_title ($fix_title) \" font \"Helvetica-Bold,14\"

    set xtics  ($x_tics)
    set xlabel  \"   $x_axis \"
    set xrange   [$x_range]

    set ytics  ($y_tics)
    set ylabel  \" $y_axis  \"
    set yrange  [$y_range]

    # position of the key
    set key left bottom

    # style of points and lines
```

```
    set pointsize 1

    ";

    # plot construct plot command
    print GNUPLOT_HEADER "plot" ;

    for(my $z_dim_index = 0; $z_dim_index < scalar(@z_dim_grep); $z_dim_index++)
    {
        #print line
        print GNUPLOT_HEADER " \"$folder_temp/$gnu_data_file\" using 1:" ,
        3*$z_dim_index +2 , ":" , 3*$z_dim_index +3,
    " title \" $z_label[$z_dim_index] \"  with yerrorlines lt $z_dim_index  linewidth 2 "
 ;

        if(($z_dim_index + 1 ) < scalar(@z_dim_grep))
            {# next data set
                print GNUPLOT_HEADER ", ";
            }
            else
            {#last data set
                print GNUPLOT_HEADER "\n";
            }
    }

    close(GNUPLOT_HEADER);

    # make graphic
    print `gnuplot $folder_temp/$gnu_header_file`;

    print `gv $folder_temp/$gnu_output_file`
}
```

Listing A.12: analysis_multiple_runs.pl

A.4. Gawk

A.4.1. tr_to_val.awk

```
# Rainer Baumann, ETH Zuerich 2004
# Master Thesis
#
# awk converter for ns-2 trace files
# from new format to val format
#

# [scheduling start]
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
# - add PING
# - add INFO PACKET
# - fix node 0 BUG
# [scheduling end]

{
    # preparations
    delete array;

    # reading line into array
    array["event"] = $1;
    for(i=2; i<NF; i++)
    {
        if ( index($i,"-") == 1 )
        {
            array[substr($i,2)] = $(i+1)
        }
    }

    # making conversion
    if ( array["Md"] == "ffffffff" )    array["Md"]="f";
    if ( array["Mt"] == "800" )         array["Mt"]="8";
    if ( array["Po"] == "REPLY" )       array["Po"]="REP";
    if ( array["Po"] == "REQUEST" )     array["Po"]="REQ";
    if ( array["Pc"] == "REPLY" )       array["Pc"]="REP";
    if ( array["Pc"] == "ERROR" )       array["Pc"]="E";
    if ( array["Pc"] == "UNSOLICITED REPLY" ) array["Pc"]="E";  # added for AODVM
    if ( array["Pc"] == "HELLO" )       array["Pc"]="HI";
    if ( array["Pc"] == "PING" )        array["Pc"]="PI";       # [scheduling]
    if ( array["Pc"] == "REQUEST" )     array["Pc"]="REQ";
    if ( array["Nw"] == "---" )         array["Nw"]="|";
    if ( array["Nl"] == "AGT" )         array["Nl"]="AG";
    if ( array["Nl"] == "RTR" )         array["Nl"]="R";
    if ( array["It"] == "DSR" )         array["It"]="D";
    if ( array["It"] == "cbr" )         array["It"]="C";

    # processing different cases
    if (array["event"] == "M")
    {
        { printf("%s\n", $0) };
    }
    else if ( \
        (array["event"] == "s") || \
        (array["event"] == "r") || \
        (array["event"] == "d") || \
        (array["event"] == "f") )
    {
        { printf("%s %s %s %s %s %s %s %s ", \
            array["event"],
```

```
        array["t"], \
        array["Hs"], \
        array["Hd"], \
        array["Nx"], \
        array["Ny"], \
        array["Nl"], \
        array["Nw"])
    };
    # mac layer extensions
    { printf("%s %s %s %s ", \
        array["Ma"], \
        array["Md"], \
        array["Ms"], \
        array["Mt"])
    };
    if ( array["Is"]~/./ )
    { printf("%s %s %s %s %s %s %s ", \
        array["Is"], \
        array["Id"], \
        array["It"], \
        array["Il"], \
        array["If"], \
        array["Ii"], \
        array["Iv"])
    };

    if ( array["P"] == "dsr" )
    { printf("%s %s %s %s %s %s>%s %s %s %s>%s ", \
        array["Ph"], \
        array["Pq"], \
        array["Ps"], \
        array["Pp"], \
        array["Pn"], \
        array["Pl"], \
        array["Pe"], \
        array["Pw"], \
        array["Pm"], \
        array["Pc"], \
        array["Pb"])
    };


    if ( array["P"] == "mpdsr" )
    { printf("%s %s %s %s %s %s>%s %s %s %s>%s ", \
        array["Ph"], \
        array["Pq"], \
        array["Ps"], \
        array["Pp"], \
        array["Pn"], \
        array["Pl"], \
        array["Pe"], \
        array["Pw"], \
        array["Pm"], \
        array["Pc"], \
        array["Pb"])
    };

    if ( array["P"] == "aodv")
    {
        if (array["Pc"] == "REQ")
        {
            printf("%s %s %s %s %s %s %s REQ ", \
                array["Pt"], \
                array["Ph"], \
                array["Pb"], \
                array["Pd"], \
                array["Pds"], \
```

```
            array["Ps"], \
            array["Pss"])
    }
    # [scheduling] added PI
    else if ( (array["Pc"] == "REP") || (array["Pc"] == "E") || \
          (array["Pc"] == "HI") || (array["Pc"] == "PI") )
    {
        printf("%s %s %s %s %s %s ", \
            array["Pt"], \
            array["Ph"], \
            array["Pd"], \
            array["Pds"], \
            array["Pl"], \
            array["Pc"])
    }
    else
        printf("Unexpected Pc \"%s\" in aodv event at line %i\n", array["Pc"], NR)
}
if ( array["P"] == "aodvm")
{
    if (array["Pc"] == "REQ")
    { printf("%s %s %s %s %s %s %s REQ ", \
        array["Pt"], \
        array["Ph"], \
        array["Pb"], \
        array["Pd"], \
        array["Pds"], \
        array["Ps"], \
        array["Pss"])
    }
    else if ( (array["Pc"] == "REP") || \
          (array["Pc"] == "E") || \
          (array["Pc"] == "HI") )
    { printf("%s %s %s %s %s %s ", \
            array["Pt"], \
            array["Ph"], \
            array["Pd"], \
            array["Pds"], \
            array["Pl"], \
            array["Pc"])
    }
    else
        printf("Unexpected Pc \"%s\" in aodv event at line %i\n", array["Pc"], NR)
}

if ( array["P"] == "arp" )
{ printf("P %s %s %s %s %s ", \
    array["Po"], \
    array["Pms"], \
    array["Ps"], \
    array["Pmd"], \
    array["Pd"])
};

if ( array["Pn"] == "tcp" )
{ printf("%s %s %s %s %s ", \
    array["Pn"], \
    array["Ps"], \
    array["Pa"], \
    array["Pf"], \
    array["Po"])
};
if ( array["Pn"] == "cbr" )
{ printf("%s %s %s ", \
    array["Pi"], \
    array["Pf"], \
    array["Po"])
```

```
        };

        if ( array["P"] == "imep" )
        { printf("%s %s %s %s %s ", \
            array["P"], \
            array["Pa"], \
            array["Ph"], \
            array["Po"], \
            array["Pl"])
        };
        if ( array["P"] == "tora" )
            printf("—P tora; not jet implemented");

         printf("\n");
    }
    else if (array["event"] == "i")
    {
    # [scheduling] info packet, just copy the information
        printf("%s \n", $0)
    }
    else
    {
        printf("Unexpected event \"%s\" at beginning of line %i\n", array["event"], NR);
    }
}
```

Listing A.13: tr_to_val.awk

A.4.2. val_to_sim.awk

```
#-----------------------------------------------------------------------------
# VAL_TO_SIM.AWK       SCRIPT TO PARSE INFORMATION FROM .VAL TO .SIM
#
# Project:  Semester Thesis SS 06:
#           'Evaluation of scheduling methods over
#           multipath routing in wireless mobile Ad Hoc networks'
#
# Authors:  Regula Goenner, Dominik Schatzmann
#
#                       [gawk Script]
#
# Date:     July 2006
#
# Input:    val file
#
# Output:   sdtout or to a .sim file, contains information of the simulation for each node
#           and, if Info packets are used, information about the routing table
#
# based on an existing script, unused variables have not been verified
# original author: val
# changes to the original part are marked with [scheduling] tags
#-----------------------------------------------------------------------------


################ OLD and NEW ######################################
#-------------------------------------------
# AODV packet, $15 = AODV, send/receive REPLY/ERROR/HI actions
# original entry in the trace file:
# s -t 3.258590807 -Hs 8 -Hd 42 -Ni 8 -Nx 864.57 -Ny 629.00 -Nz 0.00 -Ne -1.000000 -Nl RTR
# -Nw ---- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 8.255 -Id 6.255 -It AODV -Il 52 -If 0 -Ii 0 -Iv 30
# -P aodv -Pt 0x4 -Ph 0 -Pd 8 -Pds 2 -Pl 10.000000 -Pc REPLY
#
# from the val file
# 1       2          3    4     5         6       7   8   9   10  11  12   13       14
# s   3.258590807   8   42   864.57   629.00   R   |   0   0   0   0   8.255   6.255
#       -t          -Hs  -Hd   -Nx       -Ny    -Nl -Nw -Ma -Md -Ms -Mt  -Is     -Id
#
#   15   16   17  18  19    20   21  22  23     24         25
# AODV   52   0   0   30   0x4   0   8   2   10.000000   REP
#  -It   -Il -If -Ii -Iv   -Pt  -Ph -Pd -Pds   -Pl       -Pc ( = REP/HI/E)
#
#             "0x%x %d %d %d %f %s ",
#             rp->rp_type,    // -Pt
#             rp->rp_hop_count,   // -Ph
#             rp->rp_dst,     // -Pd
#             rp->rp_dst_seqno,   // -Pds
#             rp->rp_lifetime,    // -Pl
#             rp->rp_type == AODVTYPE_RREP ? "REP" :
#             (rp->rp_type == AODVTYPE_RERR ? "E" :
#              "HI"));        // -Pc
#-------------------------------------------


#-------------------------------------------
# CBR packet, $15 = C
# original entry in the trace file:
# s -t 3.229420056 -Hs 6 -Hd -2 -Ni 6 -Nx 52.18 -Ny 126.69 -Nz 0.00 -Ne -1.000000 -Nl AGT
# -Nw ---- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 6.2 -Id 8.0 -It cbr -Il 512 -If 0 -Ii 0 -Iv 32
# -Pn cbr -Pi 0 -Pf 0 -Po 0
#
# from the val file
# 1       2          3    4     5         6       7   8   9   10  11  12   13    14   15  16
# s   3.229420056   6   -2   52.18   126.69   AG   |   0   0   0   0   6.2   8.0   C   512
#       -t          -Hs  -Hd   -Nx       -Ny    -Nl -Nw -Ma -Md -Ms -Mt  -Is    -Id -Ir  -Il
#
# 17   18  19  20  21  22
```

```
#  0    0    32    0    0    0
# -If -Ii -Iv  -Pi -Pf -Po

#/* ORIGINAL
#             "-Pn cbr -Pi %d -Pf %d -Po %d ",
#             rh->seqno_,
#             ch->num_forwards(),
#             ch->opt_num_forwards());
#*/
#---------------------------------------


#---------------------------------------
# AODV packet, 15 = AODV, send/recieve REQUEST actions
# original entry in the trace file:
# r -t 3.230412376 -Hs 31 -Hd -2 -Ni 31 -Nx 95.98 -Ny 211.97 -Nz 0.00 -Ne -1.000000 -Nl
# RTR  -Nw ---- -Ma 0 -Md ffffffff -Ms 6 -Mt 800 -Is 6.255 -Id  -1.255 -It AODV -Il 52 -If 0
# -Ii 0 -Iv 30 -P aodv -Pt 0x2 -Ph 0 -Pb 1 -Pd 8 -Pds  0  -Ps 6 -Pss 4 -Pc REQUEST
#
# from the val file
# 1      2             3     4      5        6       7   8    9    10    11   12   13          14
# r    3.230412376    31    -2    95.98    211.97    R   |    0    f     6    8    6.255     -1.255
#        -t           -Hs   -Hd    -Nx      -Ny     -Nl -Nw  -Ma -Md -Ms -Mt   -Is         -Id
#
#     15    16    17   18   19     20     21   22   23    24   25   26     27
#   AODV    52    0    0    30    0x2    0    1    8     0    6    4     REQ
#    -It    -Il -If -Ii -Iv    -Pt   -Ph -Pb -Pd -Pds -Ps -Pss  -Pc
#
#             "0x%x %d %d %d %d %d %d REQ ",
#             rq->rq_type,     // -Pt
#             rq->rq_hop_count,   // -Ph
#             rq->rq_bcast_id,    // -Pb
#             rq->rq_dst,      // -Pd
#             rq->rq_dst_seqno,   // -Pds
#             rq->rq_src,      // -Ps
#             rq->rq_src_seqno); // -Pss
#
#---------------------------------------


# [scheduling] ---- start
#---------------------------------------
# INFO packet: dump the paths of the routing table
# original entry in the trace file (this is not changed in the val):
# 1  2     3       4    5    6  7    8 9  10 11 12 13   14 15   16 17 18     19
# i -t 162.777591 -Hs 68 -Hd 27 -Ia 1 -Ic 1 -In 47 -Il 68 -Ih 1 -Ir 1000.000000
#
#   Hs  source of the path
#   Hd  destination of the path
#   Ia  action (add = 1, remove = -1 path)
#   Ic  total number of paths in routing table
#   In  next hop
#   Il  last hop
#   Ih  hop count along
#   Ir  rtt along this path
#
#---------------------------------------
# [scheduling] ---- end


BEGIN {
    if(M_N!= 0) MAX_NODES = M_N;

# column numbers
    Line            = 0;        # the hole line
    Time            = 2;        # the time column, was 3
```

```
    # Node information
    ThisNode         = 3;        # Hs
    NextNode         = 4;        # Hd
    ProtocolStack    = 7;        # Nl
    Xcoord           = 5;        # X coordinate
    Ycoord           = 6;        # Y coordinate of a node


  # packet information
    # common for all packet types
    ReasonWhyDropped = 8;        # Nw
    DestMAC          = 10;       # MAC address of the destination
    IP_SourceAddress = 13;       # Is
    IP_DestAddress   = 14;       # Id
    DataOrRouting    = 15;       # -P dsr or -Pn cbr or AODV
    PacketSize       = 16;       # -Il, The size of a packet in bytes
    IP_UniqueId      = 18;       # Ii

    # protocol specific information
    # CBR
    CBR_TimesFw      = 21;       # Pf only when -Pn cbr(15 = C in new), was 49
    CBR_SequenceNumber = 20;     # Pi, only with -Pn cbr(15 = C in new), was 47

    # DSR
    NodesTraversed   = 20;       # Ph only when -P dsr(15 = D in new), was 55
    RouteReqFlag     = 21;       # == 1 if it's a DSR route request packet
    RouteRepFlag     = 23;       # == 1 if it's a DSR route reply packet
    RouteErrFlag     = 26;       # == 1 if it's a DSR route error packet

    # AODV
    AODV_PktType     = 20;       # 0x2 or 0x4
    AODV_HopCount    = 21;       # number of hops traversed
    # REQUEST s/r
    AODV_BrdcastID   = 22;
    AODV_ReqDest     = 23;
    AODV_ReqDestSeqNum = 24;
    AODV_ReqSrc      = 25;
    AODV_ReqSrcSeqNum = 26;
    AODV_REQ         = 27;
    # REPLY/HELLO/ERROR
    AODV_RepDest     = 22;
    AODV_ErrDest     = 22;
    AODV_RepDestSeqNum = 23;
    AODV_RepLifeTime = 24;
    AODV_REP         = 25;
    AODV_ERR         = 25;
    # ATTENTION: AODV ERROR is a BROADCAST packet !!!!!!!!!!!!!!!

    # [scheduling] ---- start
    #HELLO/PING
    AODV_PI          = 24;
    AODV_HI          = 25;

    #info packet new

    path_time        = 3;
    path_src         = 5;
    path_dst         = 7;
    path_action      = 9;
    path_number_total = 11;
    path_hop_next    = 13;
    path_hop_last    = 15;
    path_hop_count   = 17;
    path_rtt         = 19;

    maxPathLength    = 10;
```

```
    timeInterval    = 1;    # sampling interval
    # [scheduling] ---- end

    # ARP packet
    ARP_ifArp       = 13;
    ARP_PacketType  = 14;


# variables
    # Compute variables
    accumulateFwValue = 0;
    numFwValue      = 0;

    ## NEW
    arpREPdrops     = 0;
    arpREQdrops     = 0;
    arpDrops        = 0;

    highest_packet_id = 0;          # remember the highest unique id for data packets
    lowest_packet_id = 100000;      # remember the lowest unique id for data packets

    node[0,0]       = 0;            # node[node_id, what] -- an array to store all the
                                    # events in a node

    # [scheduling] ---- start


    #routing table
    rtable[0,0,0,0] = 0;            # save all routing infos of the INFO PATH PACKET
                                    # in this table [src,dest,path-number|info,HASH]
    rtable_s_t[0,0,0,0] = 0;        #save all paths that were found during the simulation

    data_src_dest_table[0,0] = 0;   # save all dataflows in this table

    rtable_src[0,0,0] = 0;          # save the number of path that the source sees
                                    # over the time

    # global variables to store all found paths of one src dest pair
    R_T_path[0,0]   = 0;
    R_T_path_index  = 0;

    # [scheduling] ---- end


node_name["x-coord"]        = "X coordinate of the node";
node_name["y-coord"]        = "Y coordinate of the node";

############ DATA part ###############
node_name["d-s-AG"]         = "Data pkts sent(generated)";
node_name["d-s-out"]        = "The route is found, routing agent sends the data";
node_name["d-s-out-B"]      = "The route is found, routing agent sends data, in Bytes";
node_name["d-r-ad"]         = "Data pkts received at dest";     # n1;
node_name["d-r-ff"]         = "Data pkts received for forwarding";
node_name["d-f"]            = "Data pkts forwarded";
node_name["d-f-"]           = "Data forwarded in Bytes";
node_name["d-d-total"]      = "Data pkts dropped";
node_name["d-d-total-B"]    = "Data dropped total in Bytes"; #d-d-total *packet_size
    # Drops have many reasons....
    node_name["d-d-ifq-total"]   = "Data dropped by ifq total";
    node_name["d-d-ifq-ifq"]     = "Data dropped by ifq because of overflow";
    node_name["d-d-ifq-arp"]     = "Data dropped by ifq because of arp-problem";
    node_name["d-d-ifq-end"]     = "Data dropped by ifq because of end of sim";
    node_name["d-d-rtr-total"]   = "Data dropped by rtr total";
    node_name["d-d-rtr-cbk"]     = "Data dropped by rtr because of mac-call-back";
    node_name["d-d-rtr-tout"]    = "Data dropped by rtr because of time out";
    node_name["d-d-rtr-nrte"]    = "Data dropped by rtr, no route to host";
    node_name["d-d-rtr-end"]     = "Data dropped by rtr because of end of sim";
```

```
############ Routing Protocol part ###############
node_name["p-s-total"]      = "Total number of routing packets sent (check sum)";
node_name["p-r-total"]      = "Total number of routing packets received (check sum)";
node_name["p-f-total"]      = "Total number of routing packets forwarded";
node_name["p-d-total"]      = "Total number of routing packets dropped (check sum)";

    # Drops have many reasons....
    node_name["p-d-ifq-total"]   = "Routing pkts dropped by ifq total";
    node_name["p-d-ifq-ifq"]     = "Routing pkts dropped by ifq, overflow";
    node_name["p-d-ifq-arp"]     = "Routing pkts dropped by ifq, arp-problem";
    node_name["p-d-ifq-end"]     = "Routing pkts dropped by ifq, end of sim";
    node_name["p-d-rtr-total"]   = "Routing pkts dropped by rtr total";
    node_name["p-d-rtr-cbk"]     = "Routing pkts dropped by rtr, mac-call-back";
    node_name["p-d-rtr-tout"]    = "Routing pkts dropped by rtr, time out";
    node_name["p-d-rtr-nrte"]    = "Routing pkts dropped by rtr, no route to dst";
    node_name["p-d-rtr-end"]     = "Routing pkts dropped by rtr, end of sim";
    node_name["p-d-rtr-ttl"]     = "Routing pkts dropped by rtr, ttl = 0 ";

node_name["preq-s"]         = "RREQ pkts sent";
node_name["preq-s-B"]       = "RREQ sent in Bytes";
node_name["preq-r-tot"]     = "RREQ pkts received in total";
node_name["preq-r-ad-unq"]  = "RREQ pkts received at dest (unique)";
node_name["preq-r-ad-all"]  = "RREQ pkts received at dest (total)";
node_name["preq-r-ff"]      = "RREQ pkts received for forwarding or as overhead";
node_name["preq-f"]         = "RREQ pkts forwarded";
node_name["preq-f-B"]       = "RREQ forwarded in Bytes";
node_name["preq-d"]         = "RREQ pkts dropped";

node_name["prep-s-B"]       = "RREP sent in Bytes";
node_name["prep-s-fd"]      = "RREP pkts sent from the destination of the RREQ";
node_name["prep-s-fi"]      = "RREP pkts sent from the intermediate node \
that knows the path to the destination of the RREQ";
node_name["prep-r-ad"]      = "RREP pkts received at dest";
node_name["prep-r-ff"]      = "RREP pkts received for forwarding";
node_name["prep-f"]         = "RREP pkts forwarded";
node_name["prep-f-B"]       = "RREP forwarded in Bytes";
node_name["prep-d"]         = "RREP pkts dropped";

node_name["perr-s"]         = "RERR pkts sent";
node_name["perr-s-B"]       = "RERR sent in Bytes";
node_name["perr-r"]         = "RERR pkts received";
# AODV RERR is a broadcast by default
node_name["perr-f"]         = "RERR pkts forwarded";
node_name["perr-f-B"]       = "RERR forwarded in Bytes";
node_name["perr-d"]         = "RERR pkts dropped";
# [scheduling] ---- start
node_name["ppng-s"]         = "PING sent";
node_name["ppng-r"]         = "PING pkts received";
node_name["ppng-f"]         = "PING pkts forwarded";
node_name["ppng-d"]         = "PING pkts dropped";

node_name["phel-s"]         = "HELLO sent";
node_name["phel-r"]         = "HELLO pkts received";
node_name["phel-f"]         = "HELLO pkts forwarded";
node_name["phel-d"]         = "HELLO pkts dropped";
# [scheduling] ---- end

# ..ARP must be added. #
node_name["arp-d-total"]    = "arp-d-total";
node_name["preq-d-arp"]     = "preq-d-arp";
node_name["prep-d-arp"]     = "prep-d-arp";

}
##################################################################################################
# FUNCTIONS
##################################################################################################
```

```
function get_path(src, dest, way, recursion_depth, loop_a, RTST_DEBUG)
{ # get the path from all the infomations from the i packet
    # find all paths from source to destination
    # save the path in R_T_path

    RTST_DEBUG = 0; #switch debug on off

    #set way
    way = way "-" src;

    # set rec deep
    recursion_depth++;

    if(RTST_DEBUG == 1)
    {
        printf("working point in recursion: %s\n", way);
        printf("\t rec deep: %s\n", recursion_depth);
        printf("\t src: %s\n", src);
        printf("\t dest: %s\n", dest);
        printf("\t number of next: %s\n", rtable[src, dest, "info", "path number", "0"]);
    }

    if(src == dest)
    {
        # at Destination
        # Save Path in Array
        R_T_path_index++;
        R_T_path["path",R_T_path_index]=way "-D";
        R_T_path["hops",R_T_path_index]=recursion_depth;

        if(RTST_DEBUG == 1)
        {
            printf("way from %s to %s over %s with %s hops\n", src, dest, \
                R_T_path["path", R_T_path_index], R_T_path["hops", R_T_path_index]);
        }
        return;
    }

    if(recursion_depth < MAX_NODES)
    # if the recursion depth is higher than the number of nodes in the network, then
    # we can be sure that there is a routing loop
    {
        for (loop_a = 1; loop_a <= rtable[src,dest,"info", "path number","0"]; loop_a++)
        {# over all paths
            if(rtable[src, dest, rtable[src, dest, "info", "hop next", loop_a], \
                rtable[src, dest, "info","hop last",loop_a],"status"] == "u") # path up?
            {
                # recursion
                # ask the next hop ...

                get_path( rtable[src, dest, "info","hop next", loop_a], dest, way , \
                recursion_depth , 0);
            }
        }
    }
    else
    {   #recursive deep is too big ...
        # ... never ending story? ....
        printf ( "## deep max: %s routing loop ????\n",way);
    }
}

function rtable_s_t_print(node_start,node_end,l_src,l_dst,l_path)
{   # print the routing table
    # HEADER
    printf("\n");
```

```
    printf("#RT-H-S-T\t###-------------###\n");
    printf("#RT-H-S-T\t###-- RT-S-T --###\n");
    printf("#RT-H-S-T\t###-------------###\n");
    printf("\n");
    printf("#RT-H-S-T_H\t SRC\t DEST\t #PATH\t PATH\t UP\t DOWN\n");
    printf("\n");

    for(l_src = node_start; l_src < node_end; l_src++) #src
    {
        for(l_dst = node_start; l_dst < node_end; l_dst++)#dest
        {
            if (data_src_dest_table[l_src,l_dst]==1) # only source dest
            {
                if(rtable_s_t[l_src,l_dst, "info", "path number"] != "")
                {
                    for(l_path = 1; l_path <= rtable_s_t[l_src,l_dst,"info",\
                        "path number"]; l_path++)
                    {
                        printf("#RT-S-T\t");
                        printf("%s\t", l_src);
                        printf("%s\t", l_dst);
                        printf("%s\t", l_path);
                        printf("%s\t", rtable_s_t[l_src,l_dst,"info", l_path]);
                        printf("UP%s-UP\t", rtable_s_t[l_src, l_dst,\
                        rtable_s_t[l_src,l_dst,"info",l_path],"up time"]);
                        printf("DOWN%s-DOWN\t", rtable_s_t[l_src, l_dst,\
                        rtable_s_t[l_src,l_dst,"info",l_path], "down time"]);
                        printf("\n");
                    }
                }   printf("\n");
            }
        }
    }
    printf("\n\n");
}

function rtable_src_print(node_start, node_end, l_src, l_dst)
{   # print the rtable_src
    # HEADER
    printf("\n");
    printf("#RT-H-SRC\t###-------------------------###\n");
    printf("#RT-H-SRCt###-- RT-S-T of Source --###\n");
    printf("#RT-H-SRC\t###-------------------------###\n");
    printf("\n");
    printf("#RT-H-SRC\t SRC\t DEST\t MIN HOP\t #PATHS\t @TIME\n");
    printf("\n");

# DATA
    for(l_src = node_start; l_src < node_end; l_src++) #src
    {
        for(l_dst = node_start; l_dst < node_end; l_dst++)#dest
        {
            # is a data connection
            if(data_src_dest_table[l_src,l_dst] == 1)
            {
                printf("#RT-SRC\t");
                printf("%s\t", l_src);
                printf("%s\t", l_dst);
                printf("%s\t", rtable_src[l_src, l_dst, "min hop"]);
                printf("%s\t", rtable_src[l_src, l_dst, "number of paths"]);
                printf("%s\t", rtable_src[l_src, l_dst, "time"]);
                printf("\n");
            }
        }
    }
    printf("\n\n");
}
```

```
##############################################################################################
# PROCESSING OF INPUT FILE(s)
##############################################################################################
#
# If I want to calc MAC overhead, I have to add 52 bytes header to the total overhead
#

/^r/ {
if ($Time > 180)        #remove startup phase # [scheduling] ---
{
    TN = $ThisNode;
    node[TN,"x-coord"] = $Xcoord;
    node[TN,"y-coord"] = $Ycoord;

    if ($ProtocolStack == "R" )
    {
        if($DataOrRouting == "C")
        {
            if (TN != substr($IP_SourceAddress, 1,
                    index($IP_SourceAddress,".")-1))
            {
                # check, that it is not from the AG (we don't need to
                # count it here, since it's counted in /^s/ where
                # $ProtocolStack == "AG" )
                node[TN,"d-r-ff"]++;
            }
            # In AODV, (unlike DSR) the AG-level receives a data
            # packet, if this packet is destined to the  current node.
            # In DSR, R-level always receives data packets
            # and then decides whether to forward it or to send up to
            # AG.
            # That's why we need not to check here in cas of AODV,
            # that the data packet (C) received by routing agent (R)
            # is not destined to the current node
            # (it cannot be so with AODV!). Data arrivals to
            # destination we count bellow in $ProtocolStack == AG.
        }
        else if($DataOrRouting == "AODV")
        {
            node[TN,"p-r-total"]++;
            if($AODV_REQ == "REQ")
            {
                node[TN,"preq-r-tot"]++;
                ### numRecReq_AODV[$AODV_HopCount]++;
                if(TN == $AODV_ReqDest)
                {
                    if (!((TN,"preq-r-ad-unq",
                    $AODV_BrdcastID, $AODV_ReqSrc) in node))
                    {
                        node[TN,"preq-r-ad-unq"]++;

                        node[TN,"preq-r-ad-unq",
                        $AODV_BrdcastID, $AODV_ReqSrc] =1;
                    }
                    node[TN,"preq-r-ad-all"]++;

                    # free memory in array, packet reached the
                    # destination
                    # delete nodeSentRtPt[$substr
                    # ($IP_SourceAddress,1,
                    # index($IP_SourceAddress,".") -1),$];
                    # I don't do it, since, a packet can
                    # arrive to dest earlier, that a
                    # confirmation is sent
                    # from some other node
```

```
                }
                else if (!((TN,"preq-r-ff",$AODV_BrdcastID,
                    $AODV_ReqDest, $AODV_ReqSrc) in node))
                {
                    node[TN,"preq-r-ff"]++;
                    node[TN, "preq-r-ff", $AODV_BrdcastID,
                    $AODV_ReqDest, $AODV_ReqSrc] = 1;
                }
            }
            else if($AODV_REP == "REP")
            {
                if(TN == substr($IP_DestAddress ,1,
                    index($IP_DestAddress,".") -1))
                {
                    node[TN,"prep-r-ad"]++;
                }
                else node[TN,"prep-r-ff"]++; ## here (unlike
                # above) we don't have other alternative
            }
            else if($AODV_ERR == "E")
            {
                node[TN, "perr-r"]++;
                if (!((TN,"perr-r",$AODV_ErrDest) in node))
                    node[TN, "perr-r",$AODV_ErrDest] = 1;
            }
            # [scheduling] -- start
            else if($AODV_PI == "PI")
            {
                node[TN, "ppng-r"]++;
            }
            else if($AODV_HI == "HI")
            {
                node[TN, "phel-r"]++;
            }
            # [scheduling] -- end
        }
    }
    else if ($ProtocolStack == "AG" )
    {
        node[TN,"d-r-ad"]++;
        recv_time[$IP_UniqueId] = $Time;
    }
}
}

/^s/ {

if ($Time > 180)        #remove startup phase # [scheduling] --
{
    if ($ProtocolStack == "AG")
    {
        node[$ThisNode,"d-s-AG"]++;

        # pointer to the lowest_packet_id
        if ( $IP_UniqueId < lowest_packet_id) lowest_packet_id = $IP_UniqueId;
        # pointer to the highest_packet_id
        if ( $IP_UniqueId > highest_packet_id ) highest_packet_id =$IP_UniqueId;

        # save send time in array
        send_AG_time[$IP_UniqueId] = $Time;
    }
    else if ($ProtocolStack == "R")
    {
        if($DataOrRouting == "C")
        {
            node[$ThisNode,"d-s-out"]++;
            #hdrSt_CBR[i]+= 52;#       rtSent[i]+= $PacketSize;
```

```
    node[$ThisNode,"d-s-out-B"]+= $PacketSize;
    send_out_time[$IP_UniqueId] = $Time;
}
else if($DataOrRouting == "AODV")
{
    if($AODV_ERR == "E") ###???
    # Stupid implementation of AODV RERR doesn't directly
    # allow to figure out, whether it's a f- or s-event...
    {
        if (($ThisNode,"perr-r",$AODV_ErrDest) in node)
        {
            node[$ThisNode,"perr-f"]++;
            node[$ThisNode,"perr-f-B"]+= $PacketSize;
        }
        else
        {
            node[$ThisNode,"perr-s"]++;
            node[$ThisNode,"perr-s-B"]+= $PacketSize;
        }
    }

    if($AODV_HopCount == 0 )    # was $AODV_HopCount == 1 but
                    # should be ==0 //[scheduling]
    {
        node[$ThisNode,"p-s-total"]++;      # we count all
        # AODV packets originated (sent) from this
        # node.#sendsCntr_AODV++;
        if ($AODV_REQ == "REQ")
        {
            node[$ThisNode,"preq-s"]++;
            #sendsCntrReq_AODV++;
            node[$ThisNode,"preq-s-B"]+= $PacketSize;
            if (!(($ThisNode,"preq-r-ff", $AODV_BrdcastID,\
                $AODV_ReqDest, $AODV_ReqSrc) in node))

                node[$ThisNode, "preq-r-ff", $AODV_BrdcastID, $AODV_ReqDest,\
                $AODV_ReqSrc] = 1;
        }
        else if ($AODV_REP == "REP")
        {
            node[$ThisNode,"prep-s-fd"]++;
            # This node is destination of RREQ, it sends the
            # RREP sendsCntrRep_AODV++;
            node[$ThisNode,"prep-s-B"]+= $PacketSize;
        }
    }
    else
    {
        if ($AODV_REQ == "REQ")
        {
            # That's a forward event for AODV RREQ
            node[$ThisNode,"p-f-total"]++;  #fwCntr_AODV++;
            if($DestMAC == "f")
            #It's a hack, but... (for s-event $DestMAC = 0)
            {
                node[$ThisNode,"preq-f"]++;
                node[$ThisNode,"preq-f-B"]+= $PacketSize;
            }
        }
        else if ($AODV_REP == "REP")
        {
            node[$ThisNode,"prep-s-fi"]++;
            # This node is not the destination of RREQ, but it
            # knows the route.
            # So it sends the RREP with initial HopCount =
            # hops(this_node<-->dest)+1
            node[$ThisNode,"prep-s-B"]+= $PacketSize;
```

```
                }
            }
            # [scheduling] -- start
            if ($AODV_PI == "PI")
            {
                node[$ThisNode, "ppng-s"]++;
            }
            else if ($AODV_HI == "HI")
            {
                node[$ThisNode, "phel-s"]++;
            }
            # [scheduling] -- end
        }
    }

    # If something is sent(originated) by the region, then it is "s"-event,
    # otherwise it is "f"-event
}

# [scheduling] -- start
# find all the src-dst pairs for statistical reasons
srcip = substr($IP_SourceAddress,1,index($IP_SourceAddress,".")-1);
dstip = substr($IP_DestAddress,1,index($IP_DestAddress,".")-1);

# we want to know  which node is a source
if($DataOrRouting == "C")
{
    if (src[srcip] == -1)
    {
        src[srcip] = 1;
    }
}


# build source destination table to reconstruct dataflows

#activate DEBUG output
data_table_DEBUG = 0;

if ($ProtocolStack == "AG")
{
    if($DataOrRouting == "C")
    {

        if(data_src_dest_table[$ThisNode,
            substr($IP_DestAddress, 1,index($IP_DestAddress, ".")-1)] == "")
        {#first data packet

            data_src_dest_table[$ThisNode,\
                substr($IP_DestAddress, 1, index($IP_DestAddress, ".")-1)] = 1;

            if(data_table_DEBUG == 1)
            {
                printf("\n data table - SOURCE %s \t DESTINATION %s TIME %s\n",\
                $ThisNode, substr($IP_DestAddress, 1,index($IP_DestAddress,".")-1),$Time);
            }

        }
        else if (data_src_dest_table[$ThisNode,\
            substr($IP_DestAddress, 1,index($IP_DestAddress, ".")-1)]=="1")
        {#already in table
            if(data_table_DEBUG == 1)
            {
                printf("already up\n");
            }
        }
        else
```

```
        {

        }
    }
}
# [scheduling] —— end
}

/^f/ {
if ($Time > 180)        #remove startup phase # [scheduling] ——
{
    if ($DataOrRouting == "C")
    {
        node[$ThisNode,"d—f"]++;
        node[$ThisNode,"d—f—B"]+= $PacketSize;
    }
    else if ($DataOrRouting == "AODV")
    {
        node[$ThisNode,"p—f—total"]++;

        if ($AODV_REP == "REP")
        {
            node[$ThisNode,"prep—f"]++;
            node[$ThisNode,"prep—f—B"]+= $PacketSize;
        }
        ### THIS doesn't work with AODV, the f—event for RERR doesn't
        ### exist. It's done like s—event;
        ### else if ($AODV_REP == "E") node[$ThisNode,"perr—f"]++;
        #fwCntrErr_AODV++

        # [scheduling] —— start
        else if($AODV_PI == "PI")
        {
            node[$ThisNode, "ppng—f"]++;
        }
        else if($AODV_HI == "HI")
        {
            node[$ThisNode, "phel—f"]++;
        }
        # [scheduling] —— end
    }
}
}

/^d/ {
if ($Time > 180)        #remove startup phase # [scheduling] ——
{
    dropCntr++;

    ## This treats APR—drops in IFQ (special case)
    if ($ProtocolStack == "IFQ")
    {
        if($ARP_ifArp == "A")
        {
            node[$ThisNode,"arp—d—total"]++;
            if($ARP_PacketType == "REQ")
            {
                node[$ThisNode,"preq—d—arp"]++;
            }
            else if($ARP_PacketType == "REP")
            {
                node[$ThisNode,"prep—d—arp"]++;
            }
        }
    }
```

```
if ($DataOrRouting == "AODV")
{
    #rtDr[i]+= $PacketSize;
    if ($AODV_REQ == "REQ")      node[$ThisNode,"preq-d"]++;
    else if ($AODV_REP == "REP")    node[$ThisNode,"prep-d"]++;
    else if ($AODV_REP == "E")  node[$ThisNode,"perr-d"]++;
    # [scheduling] ---- start
    else if($AODV_PI == "PI")   node[$ThisNode, "ppng-d"]++;
    else if($AODV_HI == "HI")   node[$ThisNode, "phel-d"]++;
    # [scheduling] ---- end

    node[$ThisNode,"p-d-total"]++;
    dropCntr_AODV++;
    if ($ProtocolStack == "IFQ")
    {
        node[$ThisNode,"p-d-ifq-total"]++;
        if ($ReasonWhyDropped == "|")
            node[$ThisNode,"p-d-ifq-ifq"]++;
        else if ($ReasonWhyDropped == "ARP")
            node[$ThisNode,"p-d-ifq-arp"]++;
        else if ($ReasonWhyDropped == "END")
            node[$ThisNode,"p-d-ifq-end"]++;
    }
    else if ($ProtocolStack == "R")
    {
        node[$ThisNode,"p-d-rtr-total"]++;
        if ($ReasonWhyDropped == "CBK")
            node[$ThisNode,"p-d-rtr-cbk"]++;
        else if ($ReasonWhyDropped == "TOUT")
            node[$ThisNode,"p-d-rtr-tout"]++
        else if ($ReasonWhyDropped == "NRTE")
            node[$ThisNode,"p-d-rtr-nrte"]++;
        else if ($ReasonWhyDropped == "END")
            node[$ThisNode,"p-d-rtr-end"]++;
        else if ($ReasonWhyDropped == "TTL")
            node[$ThisNode,"p-d-rtr-ttl"]++;
    }
}
else if ($DataOrRouting == "C")
{
    #recv_time[$IP_UniqueId] = -1;#Invalidate the dropped data pkt's delay
    # <-we possibly don't need it
    node[$ThisNode,"d-d-total"]++;
    dropCntr_CBR++;
    if ($ProtocolStack == "IFQ")
    {
        node[$ThisNode,"d-d-ifq-total"]++;
        if ($ReasonWhyDropped == "|")
            node[$ThisNode,"d-d-ifq-ifq"]++;
        else if ($ReasonWhyDropped == "ARP")
            node[$ThisNode,"d-d-ifq-arp"]++;
        else if ($ReasonWhyDropped == "END")
            node[$ThisNode,"d-d-ifq-end"]++;
    }
    else if ($ProtocolStack == "R")
    {
        node[$ThisNode,"d-d-rtr-total"]++;
        if ($ReasonWhyDropped == "CBK")
            node[$ThisNode,"d-d-rtr-cbk"]++;
        else if ($ReasonWhyDropped == "TOUT")
            node[$ThisNode,"d-d-rtr-tout"]++;
        else if ($ReasonWhyDropped == "NRTE")
            node[$ThisNode,"d-d-rtr-nrte"]++;
        else if ($ReasonWhyDropped == "END")
            node[$ThisNode,"d-d-rtr-end"]++;
    }
}
```

```
}
}

# [scheduling] ---- start
/^i/ {

########### Routing Table based on INFO PATH PACKETS ###############
# IDEA:
#
#   1. Copy information out of the info packet
#   2. Update global routing table (RT)
#   3. If info packet is from a data connection
#       - search all path from src to dst
#       - save new/up/down paths in rt_s_t
#   4. If info packet is from a data connection
#       - save number of paths from src to dst
#           in rt_src
#   (5.) End of simulation
#       - print rt_s_t
#       - print rt_src

    # Activate Debug output for routing table
    R_T_DEBUG = 0;

    ## Fetch Data from INFO PACKETS
    R_T_time            = $path_time;
    R_T_action          = $path_action;
    R_T_src             = $path_src;
    R_T_dst             = $path_dst;
    R_T_hop_next        = $path_hop_next;
    R_T_hop_last        = $path_hop_last;

    R_T_tot_num_path    = $path_number_total;

    R_T_hop_count       = $path_hop_count;
    R_T_rtt             = $path_rtt;

    # show received info packet
    if(R_T_DEBUG == 1)
    {
        printf("\nreceived packet at : %s\n",   R_T_time);
        printf("action: %s \n",                 R_T_action);
        printf("src: %s dest:%s\n",             R_T_src,R_T_dst);
        printf("next: %s last %s \n",           R_T_hop_next, R_T_hop_last);
        printf("path_number: %s of %s \n",      R_T_path_number,R_T_tot_num_path);
        printf("number of hops to dest: %s \n", R_T_hop_count);
    }


    ###############################
    ## Update Routing Table (R_T)
    ###############################

    # rtable[R_T_src, R_T_dst, "info", "path number", "0" ] = number of paths;
    # rtable[R_T_src, R_T_dst, "info", "hop next", #path_number] = R_T_hop_next;
    # rtable[R_T_src, R_T_dst, "info", "hop last", #path_number] = R_T_hop_last;
    # rtable[R_T_src, R_T_dst, R_T_hop_next, R_T_hop_last, "status"] = (u/d);

    if (R_T_action == "1") # add path
    {
        # does path already exist?
        if(rtable[R_T_src, R_T_dst, R_T_hop_next, R_T_hop_last, "status"] == "")
        # path new
        {
            # add path
            rtable[R_T_src, R_T_dst, "info", "path number", "0" ]++;
            rtable[R_T_src, R_T_dst, "info", "hop next", rtable[R_T_src, R_T_dst, \
```

```
            "info", "path number", "o"]] = R_T_hop_next;
        rtable[R_T_src, R_T_dst, "info", "hop last", rtable[R_T_src, R_T_dst,\
            "info", "path number", "o"]] = R_T_hop_last;

        # add new entries to the routing table
        rtable[R_T_src, R_T_dst, R_T_hop_next, R_T_hop_last,\
            "hop count"] = R_T_hop_count;

        # set up flag.
        rtable[R_T_src,R_T_dst, R_T_hop_next, R_T_hop_last,"status"] = "u";

        if(R_T_DEBUG == 1)
        {
            printf("add new path (number %s) \n",rtable[R_T_src, R_T_dst,\
                "info", "path number", "o" ] );
            printf("\t src: %s dest: %s next: %s last %s \n", R_T_src, R_T_dst, \
                R_T_hop_next, R_T_hop_last);
            printf("\t hop count: %s\n",rtable[R_T_src, R_T_dst, R_T_hop_next, \
                R_T_hop_last, "hop count"]);
        }
    }
    else if(rtable[R_T_src,R_T_dst,R_T_hop_next, R_T_hop_last,"status"] == "d")
    # path was down
    {
        # set up flag
        rtable[R_T_src,R_T_dst,R_T_hop_next, R_T_hop_last,"status"] = "u";

        if(R_T_DEBUG == 1)
        {
            printf("path was down and is set active (number %s)\n",
            rtable[R_T_src, R_T_dst, "info", "path number", "o" ] );
            printf("\t src: %s dest: %s next: %s last %s\n", R_T_src, R_T_dst, \
                R_T_hop_next, R_T_hop_last);
            printf("\t hop count: %s\n",rtable[R_T_src, R_T_dst, R_T_hop_next,\
                R_T_hop_last, "hop count"]);
        }
    }
    else
    {   #error ...
        #was already up?
        if(R_T_DEBUG == 1)
        {
            printf( "path was already up ...error?...\n");
            printf("time: %s,  status %s\n",R_T_time, rtable[R_T_src, R_T_dst, \
                R_T_hop_next, R_T_hop_last,"status"]);
            printf( "src: %s dest:%s\n",R_T_src,R_T_dst);
            printf( "next: %s last %s \n",R_T_hop_next,R_T_hop_last);
        }
    }
}
else if(R_T_action == "-1")
{   # remove path

    # does path already exist?
    if(rtable[R_T_src, R_T_dst, R_T_hop_next, R_T_hop_last, "status"] == "")
    #path is new
    {
        # error set path down that does not exists ...
        if(R_T_DEBUG == 1)
        {
            printf("set path down that that does not exist...error?..\n");
            printf("time: %s,  status %s\n",R_T_time, rtable[R_T_src, R_T_dst, \
                R_T_hop_next, R_T_hop_last,"status"]);
            printf("\t src: %s dest: %s next: %s last %s\n", R_T_src, R_T_dst, \
                R_T_hop_next, R_T_hop_last);
            printf("\t hop count: %s\n",rtable[R_T_src, R_T_dst, R_T_hop_next, \
                R_T_hop_last, "hop count"]);
```

```
            }
        }
        else if(rtable[R_T_src,R_T_dst,R_T_hop_next, R_T_hop_last,"status"] == "u")
        {   #path was up
            # set down flag
            rtable[R_T_src,R_T_dst,R_T_hop_next, R_T_hop_last,"status"] = "d";

            if(R_T_DEBUG == 1)
            {
                printf("set path down (number %s) \n",rtable[R_T_src, R_T_dst, \
                    "info", "path number", "o" ] );
                printf("\t src: %s dest: %s next: %s last %s\n", R_T_src, R_T_dst,
                    R_T_hop_next, R_T_hop_last);
                printf("\t hop count: %s\n",rtable[R_T_src, R_T_dst, R_T_hop_next, \
                    R_T_hop_last, "hop count"]);
            }
        }
        else
        {   # error ...
            # path was unknown?
            # path was already down?

            if(R_T_DEBUG == 1)
            {
                printf("set path down that wasn't up ... error?..\n");

                printf("time: %s,  status %s\n",R_T_time, rtable[R_T_src, R_T_dst, \
                    R_T_hop_next, R_T_hop_last,"status"]);
                printf("\t src: %s dest: %s next: %s last %s\n", R_T_src, R_T_dst, \
                    R_T_hop_next, R_T_hop_last);
                printf("\t hop count: %s\n",rtable[R_T_src, R_T_dst, R_T_hop_next, \
                    R_T_hop_last, "hop count"]);
            }
        }
    }
    else
    {   # error ...
        # unknown action
        if(R_T_DEBUG == 1)
        {
            printf("error: unknown action %s\n",R_T_action);
        }
    }

    # Show routing table
    if(R_T_DEBUG == 1)
    #if (data_src_dest_table[R_T_src,R_T_dst]==1) # only data connections
    {
        printf( "\nrouting table for src %s – dest %s at time %s\n", R_T_src, \
            R_T_dst,R_T_time);
        printf( "\t number of paths: %s\n",rtable[R_T_src,R_T_dst,"info",\
            "path number","o"]);

        # print all paths
        for(loop_a = 1;loop_a <= rtable[R_T_src,R_T_dst,"info",\
            "path number","o"]; loop_a++)
        {
            printf( "SRC: %s\t", R_T_src);
            printf( "DEST: %s\t", R_T_dst);
            printf( "NEXT HOP: %s\t",rtable[R_T_src, R_T_dst,"info", "hop next", loop_a]);
            printf( "LAST HOP: %s\t",rtable[R_T_src, R_T_dst,"info","hop last",loop_a]);
            printf( "UP/DOWN: %s\t", rtable[R_T_src,R_T_dst, rtable[R_T_src, R_T_dst,\
                "info", "hop next", loop_a], rtable[R_T_src, R_T_dst, "info",\
                "hop last",loop_a],"status"]);
            printf( "HOP COUNT: %s\t",rtable[R_T_src, R_T_dst, rtable[R_T_src, R_T_dst,\
                "info", "hop next", loop_a], rtable[R_T_src, R_T_dst, "info", "hop last",\
                loop_a], "hop count \n"]);
```

```
    }
}

##############################
## Calculate all possible paths from SRC DEST for the NEXT_HOP for all datapaths
## And save the paths in the global array R_T_path
## Only one path should be found!
##############################

## Check if change affects a source-destination pair

if ( (data_src_dest_table[R_T_src,R_T_dst]==1) || (R_T_time < 180))
{
    # calculate all possible path from source to dest

    R_T_path_index = 0; # number of found paths

    # (jump to next hop and search !!!ONLY DEBUG !!!)

    #get_path( R_T_hop_next, R_T_dst, "S-"R_T_src, 0);

    # recursive function to search all paths and save the paths in R_T_path
    get_path( R_T_src, R_T_dst, "S", 0);


    # Theoretically it should be only one way from x to y over this next hop,
    # as we assume to have disjoint paths. In practice we find more paths but
    # the source knows only about one. The reason for the multiple path is
    # that not disjoint path are not deleted, the will only be removed by a
    # timeout.

    if(R_T_DEBUG == 1)
    {
        if(R_T_path_index == 0 )
        {# no path from src to dest was found
            if(R_T_action == -1)
            {
                # the info packet reports this fact
                # This case is OK
            }
            else if (R_T_action == 1)
            {
                # the info packet reports a new path  but this
                # path does not exists in the routing table

                # Two reasons:
                #   1. Path is broken during message passing
                #   2. Path never exists --> error

                #printf("path from source to destination not
                # found\n");
            }
        }
        else if (R_T_path_index == 1 )
        {# one path from src to dest was found
            if(R_T_action == -1)
            { # path was removed but there is still a connection

                # Two reasons:
                # 1. Path was broken and was repaired
                # during message passing
                # 2. Multiple paths are created over this hop
                # --> error

                #printf("path was removed but there is still a
                # connection between source and dest\n");
```

```
            }
            else if (R_T_action == 1)
            {
                # OK one path was found
            }
        }
        else
        {# one path from src to dest was found
            if(R_T_action == -1)
            {# path was removed but there is still a connection

                # Two reasons:
                # 1. Path was broken and was repaired
                # during message passing
                # 2. Multiple paths are created over this hop
                # --> error

                #printf("path was removed but there is still a
                # connection between source and dest\n");

            }
            else if (R_T_action == 1)
            {# the info packet reports a new path
             # but multiple paths over this hop are found

                # Two reasons:
                #
                # 1. Flooding introduced more paths at a
                # intermediate node ( a backward path of a route
                # request crossed the source destination
                # path)
                # 2. Outdated routing infos ??  NO

                # ERROR!!!

                #printf("more than one path from source to
                # destination was found \n");
            }
        }
    }

    if(R_T_DEBUG == 1)
    {
        # NORMALLY MORE THAN THE ANNOUNCED PATHS ARE FOUND!!!!

        printf( "Found %s paths for: TIME %s\t",R_T_path_index,R_T_time);
        printf( "ACTION %s\t",R_T_action);
        printf( " SRC: %s\t", R_T_src);
        printf( " DEST: %s\t", R_T_dst);
        printf( " NEXT HOP: %s\t", R_T_hop_next);
        printf( " LAST HOP: %s\t", R_T_hop_last);
        printf( "\n");

        for(loop_a = 1; loop_a <= R_T_path_index; loop_a++)
        {
            printf("\tpath nr %i : %s recursion depth: %s\n", loop_a, \
                R_T_path["path", loop_a], R_T_path["hops", R_T_path_index]);
        }
    }
}

##############################
## update rtable_s_t ( set up / down time of the paths ) for SRC DEST over
## this next hop
## !! cover all paths found and is not restricted to the advanced informations of
## the source routing table !!
##############################
```

```
if ((data_src_dest_table[R_T_src,R_T_dst]==1) || (R_T_time < 180))
{
    delete R_T_active;
    R_T_active[0] = "";

    # check if path is new or down

    for(loop_a = 1; loop_a <= R_T_path_index; loop_a++)
    {
        if(rtable_s_t[R_T_src, R_T_dst, R_T_path["path", loop_a], "status"] == "" )
        # check if path is new
        {
            # update "info"
            # number of paths
            rtable_s_t[R_T_src, R_T_dst,"info", "path number"]++;
            # set path
            rtable_s_t[R_T_src, R_T_dst,"info", rtable_s_t[R_T_src, R_T_dst, \
            "info", "path number"]] = R_T_path["path", loop_a];

            # link hash
            rtable_s_t[R_T_src,R_T_dst, R_T_path["path", loop_a],\
            "hash"] = rtable_s_t[R_T_src, R_T_dst, "info", "path number"];

            # set uptime
            rtable_s_t[R_T_src,R_T_dst, R_T_path["path", loop_a], \
            "up time"] = rtable_s_t[R_T_src, R_T_dst,\
            R_T_path["path", loop_a], "up time"] "-" R_T_time;

            # set stat flag
            rtable_s_t[R_T_src, R_T_dst, R_T_path["path", loop_a], "status"] = "u";

            # set active
            R_T_active[rtable_s_t[R_T_src,R_T_dst,R_T_path["path",loop_a],"hash"]]= 1;

            if(0)
            {   printf ("NEW: add path %s\n",
                R_T_path["path",loop_a]);

                printf("\t\tPATH TOTAL %s", rtable_s_t[R_T_src, R_T_dst,\
                    "info","path number"]);
                printf("PATH: %s ", rtable_s_t[R_T_src, R_T_dst,"info",\
                    rtable_s_t[R_T_src, R_T_dst,"info","path number"]]);
                printf("HASH: %s ", rtable_s_t[R_T_src, R_T_dst,R_T_path["path",\
                    loop_a],"hash"]);
                printf("STAT: %s ", rtable_s_t[R_T_src, R_T_dst,R_T_path["path",\
                    loop_a], "status"]);
                printf("\n");
            }
        }
        else if (rtable_s_t[R_T_src, R_T_dst, R_T_path["path", loop_a],
            "status"] == "d")
        # check if path was down
        {   # set uptime

            rtable_s_t[R_T_src, R_T_dst, R_T_path["path", loop_a],\
            "up time"] = rtable_s_t[R_T_src, R_T_dst,\
            R_T_path["path", loop_a], "up time"] "-" R_T_time;

            # set stat flag
            rtable_s_t[R_T_src, R_T_dst, R_T_path["path", loop_a], "status"] = "u";

            # set active
            R_T_active[rtable_s_t[R_T_src,R_T_dst,R_T_path["path",loop_a],"hash"]]= 1;

            if(0)
            {
```

```
                printf ("UP: set path %s up\n", R_T_path["path",loop_a]);
            }
        }
        else if (rtable_s_t[R_T_src, R_T_dst, R_T_path["path", loop_a],\
            "status"] == "u") # check path is already up
        {#path was already up
            # set active
            R_T_active[rtable_s_t [R_T_src, R_T_dst, R_T_path["path", loop_a],\
            "hash"]] = 1;
        }
        else # unknown stat
        {
            printf ("unknown stat: %s\n", rtable_s_t[R_T_src,\
            R_T_dst, R_T_path["path",loop_a],"status"]);
        }
    }

    #show active status
    if(R_T_DEBUG == 1)
    {
        printf("Status of all paths\n");
        for(loop_a = 1; loop_a <= rtable_s_t[R_T_src, R_T_dst, "info",\
            "path number"]; loop_a++)
        {
            printf("\t index: %s Path: %s Active:%s\n", loop_a, rtable_s_t[R_T_src,\
                R_T_dst, "info", loop_a], R_T_active[loop_a]);
        }
    }

    # set passive paths to 'down'
    # problem: down-path can not be found in routing table ...
    # solution: 'ALL PATH' - 'FOUND PATH' == 'DOWN'
    # we used the 'active flag' to mark all found paths

    for(loop_a=1; loop_a<=rtable_s_t[R_T_src,R_T_dst,"info", "path number"]; loop_a++)
    {
        # get next hop out of this path string
        split(rtable_s_t[R_T_src, R_T_dst, "info", loop_a], loop_array,"-");

        if(R_T_DEBUG == 1)
        {
            if(R_T_active[loop_a] == "" && loop_array[3] == R_T_hop_next)
            {
                printf("path: %s \n",rtable_s_t[R_T_src, R_T_dst,"info",loop_a]);
                printf("status: %s \n", rtable_s_t[R_T_src, R_T_dst, rtable_s_t[R_T_src,\
                    R_T_dst, "info", loop_a],"status"]);
                printf("active: %s \n",R_T_active[loop_a]);
            }
        }

        if(R_T_active[loop_a] == ""  &&  loop_array[3] == R_T_hop_next)
        {# path was not updated and path 'flows' over the
         # R_T_hop_next (otherwise it was not found because it was
         # not searched ;-))

            if (rtable_s_t[R_T_src, R_T_dst, rtable_s_t[R_T_src, R_T_dst, "info",\
                loop_a], "status"] == "u")
            {# check if path was up
                # path was 'up', set it 'down'

                # set down time
                rtable_s_t[R_T_src, R_T_dst, rtable_s_t[R_T_src,\
                R_T_dst, "info",  loop_a],"down time"] = rtable_s_t[R_T_src, R_T_dst,\
                rtable_s_t[R_T_src, R_T_dst, "info",loop_a],"down time"] "-" R_T_time;

                # set stat flag
                rtable_s_t[R_T_src, R_T_dst, rtable_s_t[R_T_src, R_T_dst, "info", \
```

```
                    loop_a], "status"] = "d";

                if(R_T_DEBUG == 1)
                {
                    printf("paths %s is set down\n", rtable_s_t[R_T_src, R_T_dst,\
                    "info", rtable_s_t[R_T_src, R_T_dst, "info","path number"]]);

                    printf( " Time: %s\t", R_T_time);
                    printf( " ACTION: %s\t", R_T_action);
                    printf( " SRC: %s\t", R_T_src);
                    printf( " DEST: %s\t", R_T_dst);
                    printf( " NEXT HOP: %s\t", R_T_hop_next);
                    printf( " LAST HOP: %s\t", R_T_hop_last);
                    printf("\n");
                }
            }
            else if (rtable_s_t[R_T_src, R_T_dst, rtable_s_t[R_T_src, R_T_dst,\
                "info",loop_a], "status"] == "d")
            {
                # path was already down
            }
            else
            {
                # error unknown status
                printf("unknown status\n");
            }
        }
    }
}

##########################
## rtable based on src information
##########################

# process the information that can be seen at the src

#       rtable_src[R_T_src, R_T_dst, "last number of paths"]
#       rtable_src[R_T_src, R_T_dst, "time"]
#       rtable_src[R_T_src, R_T_dst, "number of paths"]
#       rtable_src[R_T_src, R_T_dst, "min hop"]

    # check if number of paths changed
        # yes --> update
            # save time
            # save number of paths

        # do nothing

    #number of paths
    if(rtable_src[R_T_src, R_T_dst,"last number of paths"] == "")
    { #new

        rtable_src[R_T_src, R_T_dst, "last number of paths"] = R_T_tot_num_path;
        rtable_src[R_T_src, R_T_dst, "time"] = "T-" R_T_time;
        rtable_src[R_T_src, R_T_dst, "number of paths"] = "N-" R_T_tot_num_path;
        rtable_src[R_T_src, R_T_dst, "min hop"] = R_T_hop_count;
    }
    else
    {
        if(rtable_src[R_T_src, R_T_dst, "last number of paths"] != R_T_tot_num_path)
        {#update
            rtable_src[R_T_src, R_T_dst, "last number of paths"]=R_T_tot_num_path;
            rtable_src[R_T_src, R_T_dst, "time"] = \
                 rtable_src[R_T_src, R_T_dst, "time" ] "-" R_T_time;
            rtable_src[R_T_src, R_T_dst, "number of paths"] = \
                 rtable_src[R_T_src, R_T_dst, "number of paths"] "-" R_T_tot_num_path;
```

```
            if(R_T_DEBUG == 1)
            {
            printf("\ntime: %s \n", R_T_time);
            printf("number of paths: %s \n", R_T_tot_num_path);
            printf("time-line: %s \n", rtable_src[R_T_src, R_T_dst, "time"] );
            printf("num-line:  %s \n", rtable_src[R_T_src, R_T_dst, "number of paths"] );
            }
        }
    }

    #save min hopcount
    if(rtable_src[R_T_src, R_T_dst, "min hop"] > R_T_hop_count)
    {
        rtable_src[R_T_src, R_T_dst, "min hop"] = R_T_hop_count;
    }
}

# [scheduling] ---- end

##############################################################################################

END {
# FILENAME is built-in variable that reflects the name of the processed file

########
# substr.
# UPDATE this array, if you change node[][] & node_name[]

############ DATA part ###############
    names[-2] = "x-coord";
    names[-1] = "y-coord";
    names[0] = "d-s-AG";
    names[1] = "d-s-out";
    names[2] = "d-r-ad";
    names[3] = "d-r-ff";
    names[4] = "d-f";
    names[5] = "d-d-total";
    names[6] = "d-d-ifq-total";
    names[7] = "d-d-ifq-ifq";
    names[8] = "d-d-ifq-arp";
    names[9] = "d-d-ifq-end";
    names[10] = "d-d-rtr-total";
    names[11] = "d-d-rtr-cbk";
    names[12] = "d-d-rtr-tout";
    names[13] = "d-d-rtr-nrte";
    names[14] = "d-d-rtr-end";
############ Routing Protocol part ###############
    names[15] = "p-s-total";
    names[16] = "p-r-total";
    names[17] = "p-f-total";
    names[18] = "p-d-total";
    # Drops have many reasons....
    names[19] = "p-d-ifq-total";
    names[20] = "p-d-ifq-ifq";
    names[21] = "p-d-ifq-arp";
    names[22] = "p-d-ifq-end";
    names[23] = "p-d-rtr-total";
    names[24] = "p-d-rtr-cbk";
    names[25] = "p-d-rtr-tout";
    names[26] = "p-d-rtr-nrte";
    names[27] = "p-d-rtr-end";
    names[28] = "p-d-rtr-ttl";

    names[29] = "preq-s";
    names[30] = "preq-r-tot";
    names[31] = "preq-r-ad-unq";
    names[32] = "preq-r-ad-all";
```

```
names[33] = "preq-r-ff";
names[34] = "preq-f";
names[35] = "preq-d";

names[36] = "prep-s-fd";
names[37] = "prep-s-fi";
names[38] = "prep-r-ad";
names[39] = "prep-r-ff";
names[40] = "prep-f";
names[41] = "prep-d";

names[42] = "perr-s";
names[43] = "perr-r";
#names[41] = "perr-r-ad";
#names[42] = "perr-r-ff";
names[44] = "perr-f";
names[45] = "perr-d";

# ..ARP must be added. #
names[46] = "arp-d-total";
names[47] = "preq-d-arp";
names[48] = "prep-d-arp";

names[49] = "d-s-out-B";
names[50] = "d-f-B";
names[51] = "preq-s-B";
names[52] = "preq-f-B";
names[53] = "prep-s-B";
names[54] = "prep-f-B";
names[55] = "perr-s-B";
names[56] = "perr-f-B";

# [scheduling] ---- start
names[57] = "ppng-s";
names[58] = "ppng-r";
names[59] = "ppng-f";
names[60] = "ppng-d";

names[61] = "phel-s";
names[62] = "phel-r";
names[63] = "phel-f";
names[64] = "phel-d";
# [scheduling] ---- end

START_INDEX = -2;
MAX_TITLES = 65;

printf("Node\t");
for(names_id = START_INDEX; names_id < MAX_TITLES; names_id++)
printf("%s\t", names[names_id]);
printf("\n");
for(node_id = 0; node_id < MAX_NODES; node_id++)
{
    printf("%d\t", node_id);
    for(names_id = START_INDEX; names_id < MAX_TITLES; names_id++)
    {
        tmp = names[names_id];
        total[names_id]+= node[node_id, tmp];
        printf("%d\t",node[node_id, tmp]);
    }
    printf("\n");
}

print "TOTAL:";

printf("Node\n");
for(names_id = START_INDEX; names_id < MAX_TITLES; names_id++)
```

```
    {
        printf("%s\t", names[names_id]);
        if (names_id <65 || total[69] == 0)
        {
            printf("%s\n", total[names_id]);
        }
        else
        {
            printf("%s\n", total[names_id]/total[69]);
        }
    }

    printf("\n");
    # Average pkt delay:
    print "AVERAGE DATA PACKET DELAY";

    find_route_cntr = 0;
    travel_time_cntr = 0;
    total_delay_cntr = 0;
    for ( packet_id in send_AG_time)
    {
        find_route = send_out_time[packet_id] - send_AG_time[packet_id];
        travel_time = recv_time[packet_id] - send_out_time[packet_id];

        # We have here <, but not < = , since we wanna count only real delays
        # for unknown paths, not when the path is known.
        if ( send_AG_time[packet_id] < send_out_time[packet_id] )
        {
            avrg_find_route+= find_route;
            find_route_cntr++;
        }
        if ( send_out_time[packet_id] < recv_time[packet_id] )
        {
            avrg_travel_time+= travel_time;
            travel_time_cntr++;
        }
        if ( send_AG_time[packet_id] < recv_time[packet_id] )
        {
            avrg_total_delay+= recv_time[packet_id] - send_AG_time[packet_id];
            total_delay_cntr++;
        }
    }

printf("%s\t%s\n", "AVRG time to find a route", avrg_find_route/find_route_cntr);
printf("%s\t%s\n", "AVRG time a data packet travels", avrg_travel_time/travel_time_cntr);
printf("%s\t%s\n", "AVRG TOTAL delay of a data packet",avrg_total_delay/total_delay_cntr);
printf("%s\t%s\n", "RATIO", total[2]/total[0]);

printf("\n%s\n", "CHECK IF TRACE WAS COMPLETE");
printf("%s\t%s\t%s\t%s\n", "first data with id", lowest_packet_id ," packet was sent at",\
    send_AG_time[lowest_packet_id]);
printf("%s\t%s\t%s\t%s\n", "last data with id", highest_packet_id ," packet was sent at",\
    send_AG_time[highest_packet_id]);
printf("last timestamp \t%s\n", $Time);

# [scheduling] ---- start
rtable_s_t_print(0,81);
rtable_src_print(0,81);
# [scheduling] ---- end

}
```

Listing A.14: val_to_sim.awk