Simulations on Multipath Routing Based on Source Routing

Bachelorarbeit
Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Abdalla Hassan

2008

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Table of Contents

Acknowledgment

My sincere thanks to Thomas Staub of University of Bern for being helpful tutor,
to my brother Nasir and his family for their supporting.

Abstract

In recent years, on-demand routing protocols have attained more attention in mobile ad hoc networks as compared to other routing schemes due to their abilities and efficiency. They are able to organize themselves dynamically with lower memory overhead and lower bandwidth requirement than table driven protocols (proactive protocols). However, as there were still some bottlenecks in the pioneering versions of on-demand routing protocols, more research work has been done to rectify most of these problems. For example, many on-demand routing protocols, such as Associativity Based Routing (ABR) protocol, use a single route per data session. Therefore a new route discovery has to be initiated if the active route is broken. Therefore huge delays in the data session occur. Also, many on-demand routing protocols, such as Dynamic Source Routing (DSR) protocol, do not support Quality-of-Service (QoS). This may decrease the quality of multimedia transmissions.

Several protocols have been developed to address these challenges. To resolve the single route per data session problem, Ad hoc On-Demand Distance Vector Multipath AODVM-R [19] protocol uses multiple routes maintained at the intermediate nodes. Whereas the Split Multipath Routing (SMR) [4] protocol uses multiple routes maintained from the source node for each data session. To address the Quality-of-Service (QoS) problem, protocols, such as Multipath Dynamic Source Routing (MP-DSR) [5] protocol, supports QoS by considering the end-to-end reliability of the individual paths.

During this Bachelor thesis, we implemented and evaluated two on-demand routing protocols, namely Dynamic Source Routing (DSR) [1, 2, 3] and the Split Multipath Routing (SMR) [4]. The thesis is structured as follows. In Chapter 1, mobile ad hoc networks are shortly presented. Chapter 2 discusses different on-demand ad hoc routing protocols. In Chapters 3 and 4 our implementation and integration of the DSR and SMR protocols with OMNeT++ and INET framework is described. Chapters 5 and 6 present our evaluations and conclusions.

1. INTRODUCTION

MANET

A mobile ad hoc network (MANET) is a collection of mobile hosts forming a temporary network without the aid of any established infrastructure, or support of any base station. A MANET has no central administration point [9]. All the hosts work at the same time as routers and communicate with each other over wireless connections. The nodes may also be mobile; they can move freely, and organize themselves randomly i.e. each host can dynamically enter and leave the network. Thus, the network topology may change frequently and rapidly. This means that the network has to adapt itself to the current topology. A MANET may either work as a self-configured stand-alone network or may be connected to the Internet through gateway nodes (*Figure 1.1)*.

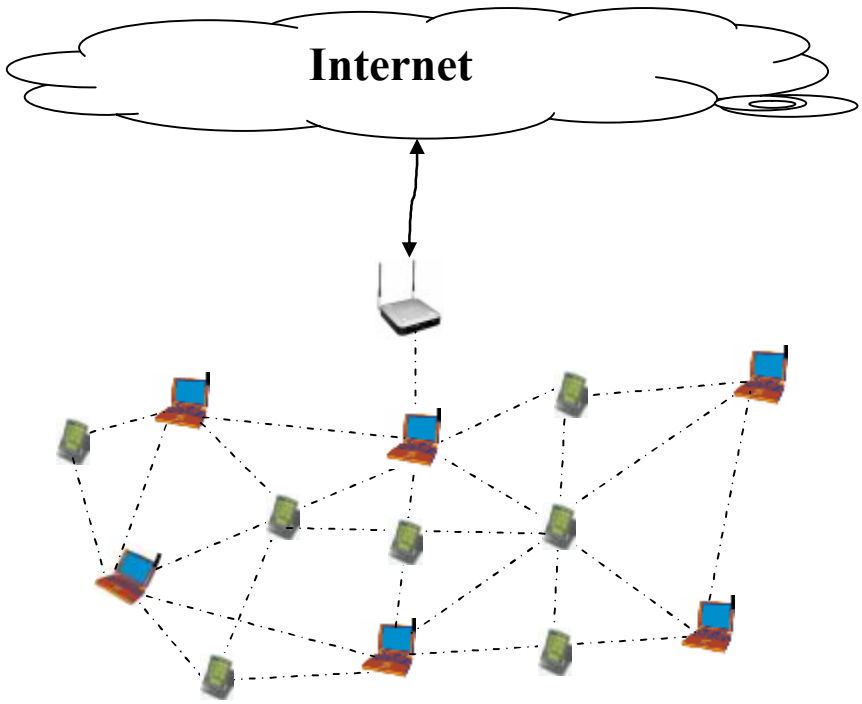


Figure 1.1 MANET

In the first part of this thesis, we present protocols which support unipath traffic from the source node to the destination node, where the focus is on Dynamic Source Routing (DSR) protocol. The second part presents multipath routing protocols, where Split Multipath Routing (SMR) protocol, which is based on DSR, is introduced.

The remainder of this Bachelor thesis is organized as follows. In Chapter 2, the related work is presented with more details about our researched protocols. In Chapter 3, the implementation of DSR and SMR protocols in OMNeT++ simulator are presented. In Chapter 4, the integration steps and arised problems in INET framework are shown. In Chapter 5, the simulation and evaluation of DSR and SMR are presented. Finally, conclusions and remarks are made in Chapter 6.

Chapter 2: RELATED WORK

2.1 MANET - Routing protocol

Routing protocols for MANETs can be categorized in various ways. They can be classified as proactive and reactive routing depending on several factors. Such factors can be for example the time taken for routes discovery or routing information update mechanism. *Figure 2.1* presents some routing protocols for MANETs.

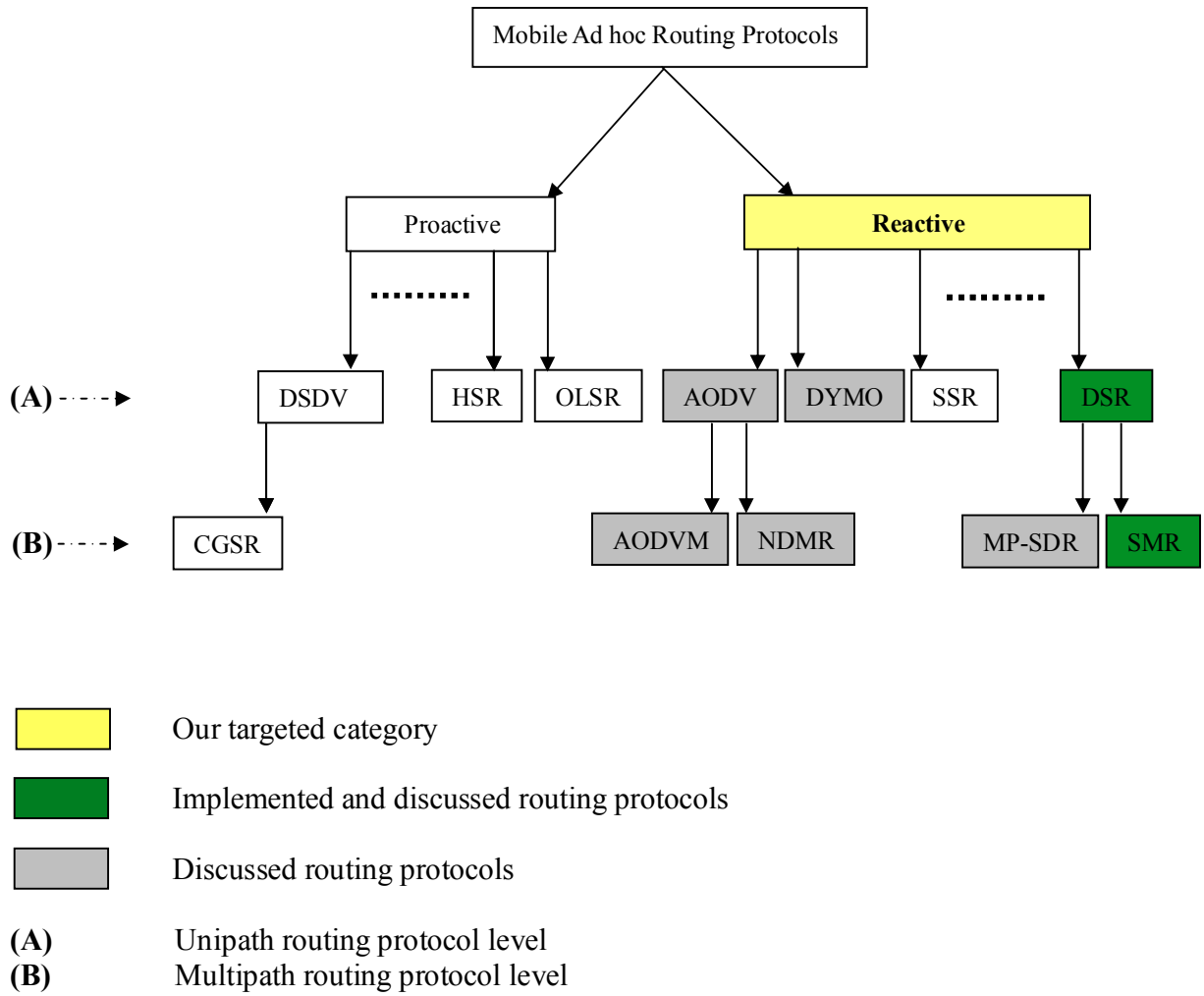


Figure 2.1: Ad hoc routing protocols

In proactive routing, every host maintains at least one routing table to represent the whole topology of the network. The tables (of each host) are updated continuously. Therefore, routes are already available at any time some hosts want to communicate with each other.

In order to maintain up-to-date routing information at all hosts, topology information has to be exchanged between all hosts on a regular basis. This increases the overhead in the network. On one hand, substantial bandwidth is used for the large control traffic; on the other hand, routes are always available in shortly for any communication request. This reduces the delays of data transmissions. One of the most important proactive protocols is the Optimized Link State Routing protocol (OLSR) [18].

Unlike proactive routing protocols, reactive routing protocols initiate a route discovery process when needed. This reduces the overhead as compared to proactive routing protocols, but it increases the transmission delay.

Another classification can be made according to number of paths a routing protocols delivers per source destination pair. There exist unipath and multipath routing protocols. Unipath routing protocol: one route is used to deliver data from source node to destination node. Multipath routing protocol: more than one route is used to deliver the data.

In the next section, the unipath routing protocols are discussed.

2.2 Unipath Routing Protocols

There exist two major classes of on-demand routing algorithms, namely distance vector and source routing. A distance vector algorithm uses some similar features as the Bellman-Ford algorithm to calculate the routing paths. It requires that a node informs its neighbors periodically. The packets include the next-hop in their header and each intermediate node adapts this information accordingly along the path. A source routing algorithm requires that a node knows the complete paths to the destination.

2.2.1 AODV - Ad hoc On-Demand Distance Vector [6, 7]

AODV protocol is defined by the RFC 3561, written by Charles Perkins and Elizabeth [6]. AODV has some similar features as the Bellman-Ford distant vector algorithm, but it has been improved to work in a mobile environment [7]. AODV uses hop-by-hop routing (*AODV Route Discovery Process - Figure 2.3)*. Every node forwards data packets towards a destination node according to its routing table (*Figure 2.2)*. The routes in the AODV routing table are kept up to date as long as they are needed by the source. AODV maintains a single path per a destination. The routing is divided into two basic mechanisms. The first one is the route discovery. It is responsible for finding a route to the destination if none is currently available in the routing table of the node. The second one is the route maintenance which keeps the routes up-to-date, e.g. removes broken paths.

AODV protocol only works in a network where the communication links are bidirectional because if an (intermediate) node receives either a Route REQuest (RREQ) packet or a Route REPly (RREP) packet, it caches the previous node in its routing table (*figure* 2.2) as a next hop to the end nodes.

Destination Id	Next hop

Figure 2.2: AODV simple routing table

2.2.1.1 AODV Protocol - Control Packets

AODV uses four types of routing messages. They are explained as follows:

- RREQ
  If a node wants to communicate with other node but no route is available, the source node starts a route discovery by broadcasting a Route REQuest (RREQ) message in the network.
- RREP
  If it is a destination node or an intermediate node has a valid route to the desired destination, it replies to a RREQ by unicasting a Route REPly (RREP) message back to the source node.

- RERR
  If a path breaks, the intermediate node generates a Route ERRor (RERR) message to inform its end nodes of the occurred link break.
- HELLO
  Each node broadcasts periodically a message with time to live (TTL) = 1, in order to maintain its neighbour list.

2.2.1.2 AODV Route Discovery

If a source has no entry for a destination in its routing cache, it starts a route discovery process. It floods a RREQ packet in the network. The RREQ includes header fields with the following parameters: request ID, source node ID, destination node ID, hop count, sequence number of the source node, sequence number of the destination node and TTL (time-to-live). If an intermediate node receives a RREQ packet, it checks if it is the destination node. If not, it checks if it has seen this RREQ before by checking the request ID and source node ID. If this is the case the node just drops the packet and does not forward the RREQ any further. This avoids loops in the route. If the RREQ packet is not dropped, the intermediate node searches in its route cache table. If there is an active route to the destination, it sends back a RREP with its route entity. Otherwise it just rebroadcasts the received RREQ. If the destination node has received the RREQ, it generates a RREP packet and sends it back in reverse way to the source.

If an intermediate node receives either a RREQ or a RREP packet, it stores information about the previous node from which the packet was received in its routing table (*Figure 2.2*). With this mechanism, hop-by-hop routing, a node can therefore decide which next hop it can use to reach a destination node. [13]

2.2.1.3 AODV Route Maintenance

If a node tries to forward a message, but detects that there is a link break, i.e., the next node is not more reachable, the forwarding node sends back a RERR message towards the source node. Whenever a node receives a RERR message, it deletes all routes containing this broken link in its routing table. When the source receives the RERR packet, it also updates its routing table, but it does not send the RERR packet anywhere. If the data session has not yet been completed and the source does not have any other route to the destination, the node starts the route discovery process.

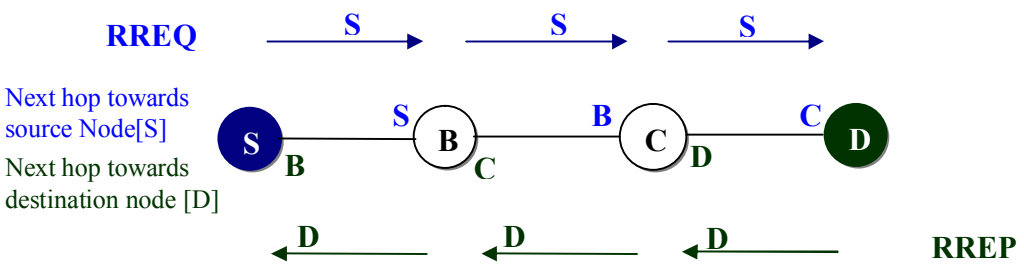


Figure 2.3: AODV Route Discovery Process

2.2.1.4 AODV Discussion

AODV mechanisms are not complex to implement. The header of data packets includes only the next-hop information and therefore requires less overhead than in source routing.

On the other hand, one of the major problems of AODV is the maintenance of only one route per a destination. This means that every time a path is broken, AODV has to initiate a route discovery process, which leads to more overhead in the network.

2.2.2 DYMO - Dynamic MANET On-Demand [11, 12]

The DYMO routing protocol is the next generation of AODV protocol. It also uses hop-by-hop routing mechanism. However, DYMO adds some new features like the path accumulation mechanism, which means that each node appends its own address to the routing packets (*Figure 2.4)*. DYMO protocol uses the same two basic operations like AODV, route discovery and route maintenance. Routes are discovered on-demand when a node needs to send data to a destination currently not in its routing table. The source node broadcasts a RREQ packet. If the RREQ reaches its destination, a RREP is sent back containing the discovered accumulated path. DYMO defines the same messaging types as AODV protocol – RREQ, RREP, RERR and HELLO message. DYMO has achieved some improvements from the path accumulation mechanism such as: a reduced number of routing control packets, especially RREQ packets (i.e. less overhead) and a much quicker topology discovery. However, the path accumulation feature increases the packet size. Therefore, the benefit may be reduced in case that the node does not use the new discovered routes.

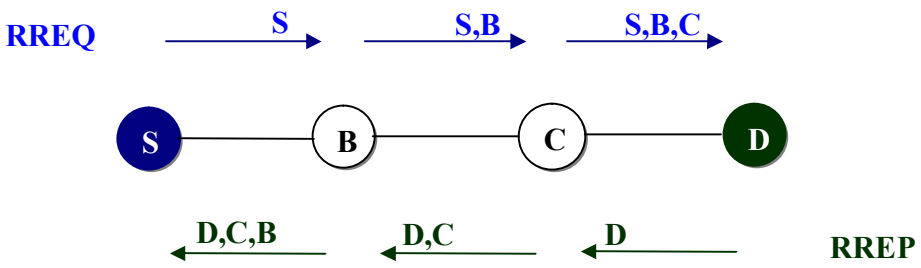


Figure 2.4 DYMO path accumulation in route discovery process

2.2.3 DSR Protocol- Dynamic Source Routing [1, 2, 3a, 3b]

Unlike AODV and DYMO, DSR is a source routing protocol. This means that the source node adds the whole route up to the destination node to the packets header (*Figure 2.5)*. As this is the case with most reactive ad hoc routing protocols, DSR is based on the two basic mechanisms namely route discovery and route maintenance. During the route discovery a route is set up on-demand. The route maintenance monitors an established connection during a communication between nodes [2].

DSR is able to operate on networks containing unidirectional links but it works optimal in a network with bidirectional links.

Option Type	Option Data Length	Identifications
Target Address		
Address[1]		
Address[2]		
….		
Address[n]		

Figure 2.5: DSR data packet header format

2.2.3.1. DSR Basic Functions

If a source node originates a new data packet to some destination node, it adds the whole path "source route" in the packet header. The source node searches for a route to that destination in its own route cache table (*Figure 2.6)*. If it does not find an entry, it initiates a route discovery process to dynamically find a route to the destination node. Route discovery is similar to that in AODV. However, there are some differences.

First, the RREQ packet (*Figure 2.7)* broadcasted by a source nodes include a new field, the route record, which saves the nodes the RREQ packet traverses on it travels towards the destination node. Second, intermediate nodes receiving the RREQ check if their address is included in the route record. Third, if the RREQ packet arrives at the destination node, it checks its route cache for another route to the source node. If it finds one, the destination node generates a RREP packet (*Figure 2.8)*, adds the route record (from RREQ packet) and sends the RREP back to the source node over its own route. Therefore DSR can work in a unidirectional link field where the revise route is not available and using other routes for replying to the RREQs. Otherwise, the destination node sends the RREP packet over the reverse route back to the source node. The route maintenance is processed as mentioned in AODV. If an intermediate node could not forward a data packet, it generates a RERR packet (*Figure 2.9)* and sends it back to the source node. Whenever a node receives a RERR message, it deletes all its routes containing the broken link.

Destinations	**Source Route Record**				**…**
Destination[i]	Address[1]	Address[2]	…..	Address of Dest[i]	…

Figure 2.6 DSR route cache table packet format

Option Type	Option Data Length	Identifications
Target Address		
Address[1]		
Address[2]		
….		
Address[n]		

Figure 2.7 DSR RREQ packet format

Option Type	Opt Data Length	L| Reserved
Address[1]		
Address[2]		
….		
Address[n]		

Figure 2.8 DSR RREP packet format

Option Type	Opt Data Length	Error Type	Reservd|Salvage
Error Source Address			
Error Destination Address			
Type-Specific Information			

Figure 2.9 DSR RERR packet format

2.2.3.2. Additional Features [1, 2]

In addition to the two basic mechanisms mentioned above, DSR protocol provides further features. These features make the DSR more efficient but could also cause some challenges as will be mentioned later in the discussion section.

- Route Discovery Features

  - **Caching Overheard Routing Information**
    If a node is forwarding or overhearing any routing packet it updates its own route cache.

  - **Replying to Route Requests Using Cached Routes**
    If an intermediate node receives a RREQ packet to other destination and has a valid route to the requested destination, the intermediate node unicasts a RREP packet back to the source node.

  - **Route Request Hop Limits**
    Each RREQ packet contains a "hop limit" or time-to-live “TTL” field in its IP header. The TTL is used to limit the propagation of the RREQ packet with the aim to reduce the routing control packets overhead on the network.

- Route Maintenance Features

  - **Packet Salvaging**
    Packet salvaging occurs if an intermediate node forwarding a data packet detects that the link to the next node is broken and it has another valid route to the destination in its route cache. Otherwise, the node drops the data packet. In all cases, the node sends back a RERR packet toward the source node.

  - **Automatic Route Shortening**
    If a node is able to overhear a packet carrying a source route, which will come to it later, the node should send back a RREP with the shorter path to the source node. For example: in the *Figure 2.10*, where node[d] can overhear the packet when node[b] transmits it to node[c], node[d] returns a RREP with [a-b-d] route source to node[a].

  - **Increased Spreading of Route Error Messages**
    If a source node receives a RERR, it propagates this RERR to its neighbors by including it in its next RREQ. In this way, the source node does not respond with a new RREP contain the same invalid link.

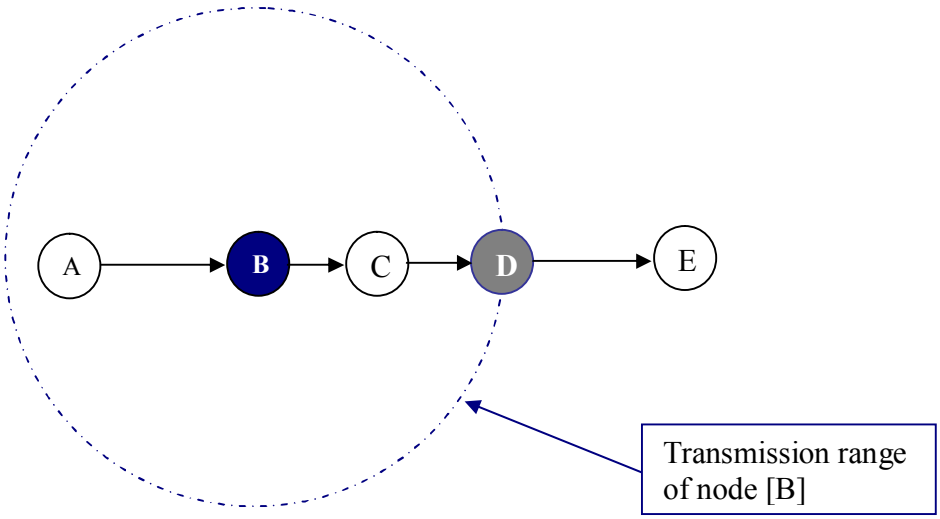


Figure 2.10: Automatic route shortening

2.2.3.3. DSR Discussion

DSR adapts quickly to routing changes. Since a node can overhear routing information and therefore can have more than one route to a destination, there is lower overhead on the network as compared to AODV. The routes used are loop-free. On the other hand, the header size on RREQ packet is not consistent, since the intermediate nodes append their IDs in the route record. Routing packets could carry wrong information. In this case "caching overheard routing information" mechanism could produce other problems. For example a node uses a wrong path to send data or replies with invalid route information to other RREQs.

In this section, two types of routing algorithms have been discussed, namely the distance vector and the source routing algorithms. To elaborate these two algorithms, the three most common unipath ad hoc network routing protocols have also been presented. These are AODV, DYMO and DSR.

In the next section, we discuss multiple paths routing protocols, which can be useful in improving the effective bandwidth of communication pairs, responding to congestion and burst traffic, and therefore increases the delivery reliability.

2.3. Multipath Routing Protocols

AODV and DSR are very similar, but AODV mechanisms are easier to implement and to integrate with other mechanisms using other different routing protocols. Moreover, AODV has better scalability and its header size on data packet is relative constant. However, AODV maintains only one route per destination. This is one of the major problems in AODV, since every time a route is broken; a route discovery has to be initiated. This leads to more overhead, higher delays and high packet lost.

On the other hand, DSR seems to be more stable and has less overhead than AODV. DSR can make use of multiple paths and does not send a periodic packet as AODV. Moreover, it stores all usable routing information extracted from overhearing packets. However, these overheard route information could lead to inconsistencies.

In order to alleviate the above-mentioned problems two multipath routing protocols (AODVM and NDMR) based on AODV protocol as well as other two multipath routing protocols (SMR and MP-DSR) based on DSR protocol are introduced.

2.3.1 AODVM Protocol - Ad-hoc Over Distance Vector Multipath [13, 14]

AODVM is a multipath protocol which is based on the AODV protocol. It is able to detect and maintain multiple node-disjoint paths for each destination. The main idea is that the source nodes are responsible for maintaining the alternate routes to a destination. But the intermediate nodes maintain at most one forwarding table per flow. In addition to the routing table in AODV *(Figure 2.11)* there exists a RREQ table in AODVM *(Figure 2.12)*.

Destination	Source	Last Hop	Next Hop

Figure 2.11 A simple AODVM routing table

Destination	Source	Neighbor who transmitted the RREQ (received hop)	Hops to Source	Expiration Timer

Figure 2.12 A simple AODVM RREQ table

2.3.1.1. AODVM Routing

In order to find multiple paths, intermediate nodes have to forward the duplicated RREQ packets - unlike in AODV. Every time an intermediate node receives a duplicated RREQ packet it adds the routing information in its RREQ table as a new record and rebroadcasts the RREQ packet. If the destination receives the first RREQ packet, it updates its sequence number and sends a new RREP back over the same path in the reverse direction. The RREP contains an additional field called "last hop id" compared to AODV RREP. The same is done with the received duplicate RREQs. If an intermediate node receives a RREP packet, it adds a route entry to its routing table and forwards the RREP packet to a neighbor which in its RREQ table and has the shortest path to the source node. Then the record about this neighbor is deleted from its RREQ table. However, the intermediate nodes are not allowed to return a RREP packet to the source directly since we want to guarantee node-disjoint routes and to get as many as possible paths.

2.3.2. NDMR Protocol - Node Disjoint Multipath Routing [15, 16, 17]

Xuefei Li and Laurie Cuthbert are the authors of NDMR protocol [15, 16, 17]. NDMR extends the AODV protocol with new features such as the path accumulation and reverse-route-table. The path accumulation feature is similar to the one of DSR. If a (intermediate) node receives a RREQ packet, it appends its own address, after that the destination node selects node-disjoint paths *(Figure 2.15)*. This results in a much lower overhead and in multiple node-disjoint paths *(Figure 2.16, 2.17)*. The second new feature is reverse-route-table. A reverse route table basically is similar to the routing table, however with the other direction.

2.3.2.1. NDMR Routing

In order to detect multiple node-disjoint paths with a low routing overhead, NDMR defines a new field in the RREQ packet. If the RREQ packets are generated or forwarded by the nodes, every node has to append its own ID to the RREQ packet. If an intermediate node receives a RREQ packet, it checks the hop count of the RREQ packet with respect to the "too-large-hop-count-rule". This means that the hop count of the duplicated RREQ packet is not larger than the hop count of the first RREQ packet. If the RREQ packet has an acceptable count, the intermediate node adds itself to the RREQ packet and rebroadcasts it; otherwise, the RREQ packet is discarded.

After the RREQ packets have arrived at the destination node, the destination node decides which routes are node-disjoint *(Figure 2.16)*, adds them in the reverse routing table and replies with a RREP. Similar to the RREQ process, RREP has a record field, and the intermediate node may use the information in this field. This may lead to a reduced overhead in the network (*Figure 2.15)*. For example, there are less route discovery processes required.

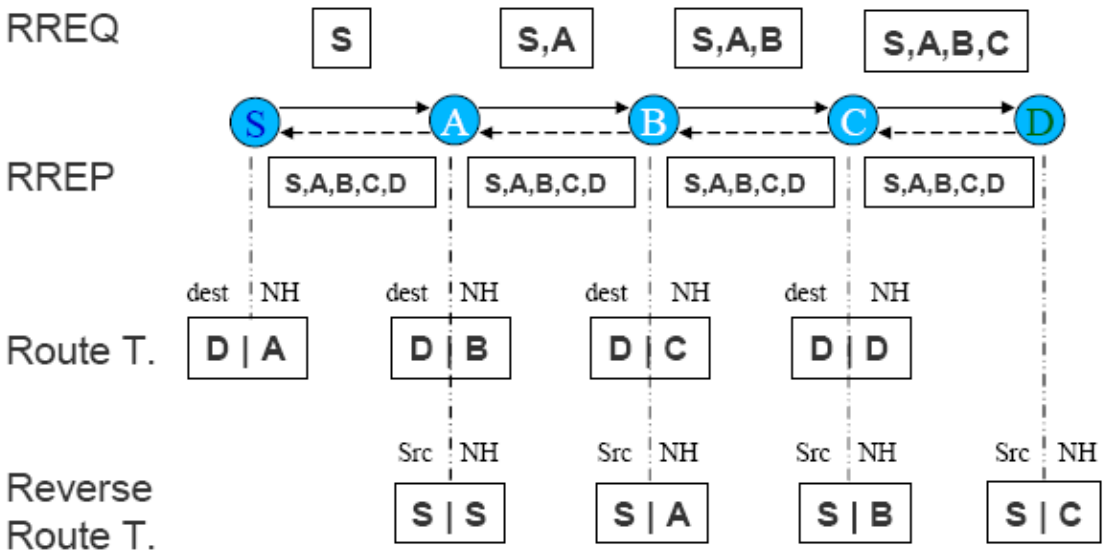


Figure 2.15 Path Accumulation in NDMR

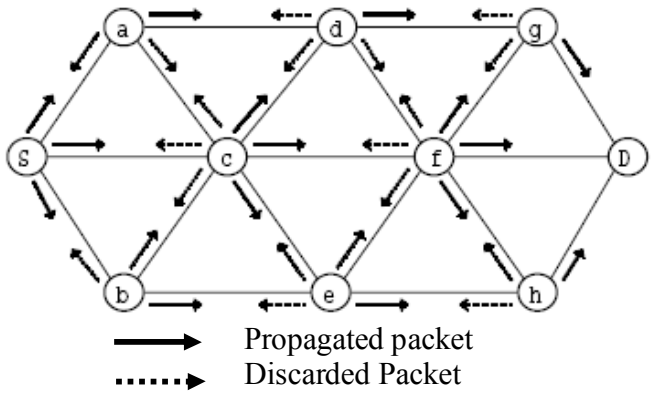


Figure 2.16 RREQ process in NDMR [17]

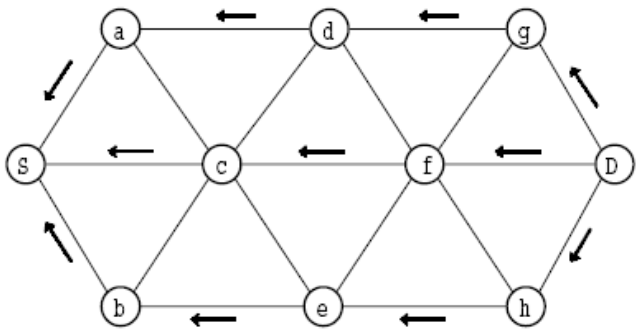


Figure 2.17 Node-Disjoint RREP Paths in NDMR [17]

2.3.3. SMR Protocol - Split Multipath Routing [4]

Split Multipath Routing (SMR) protocol is written by Sung-Ju Lee and Mario Gerla [4]. It works on the basis of DSR, i.e. it uses the same source routing mechanism. It constructs its multiple routes on-demand. The shortest delay path is one of these routes. The other paths are maximal disjoint according to this first one. To avoid delays, traffic jams and to use the network resources efficiently, data traffic is distributed onto these multiple routes.

2.3.3.1. SMR Routing:

SMR uses one route discovery process to accumulate as many as possible routes to the destination node. This route discovery process runs in the same way as in DSR. However, there are more steps involved in processing RREQ packets at intermediate and destination nodes. If an intermediate node receives a RREQ packet, it adds its own address and rebroadcasts the RREQ packet. Whenever an intermediate node receives another RREQ from the same source node and with the same request ID, i.e. a duplicated RREQ, the node checks the following two things *(Figure 2.18.a)*. First, the RREQ packets are checked if they traversed through different incoming link. Second, the hop count (of the RREQ) is checked if it is not larger than that of the first received RREQ. Then the node appends its own ID and forwards the RREQ packets. Otherwise the RREQ packet is discarded. Additionally, intermediate nodes are not allowed to reply directly with a RREP on a RREQ packet.

If the first RREQ packet arrives at the destination node, a RREP is generated and sent back on the reverse path which is the "Shortest delay path". Then the destination node waits a period of time and selects multiple disjoint routes, according to the first path, and sends RREP packets back to the source via the selected routes *(Figure 2.18.b)*.

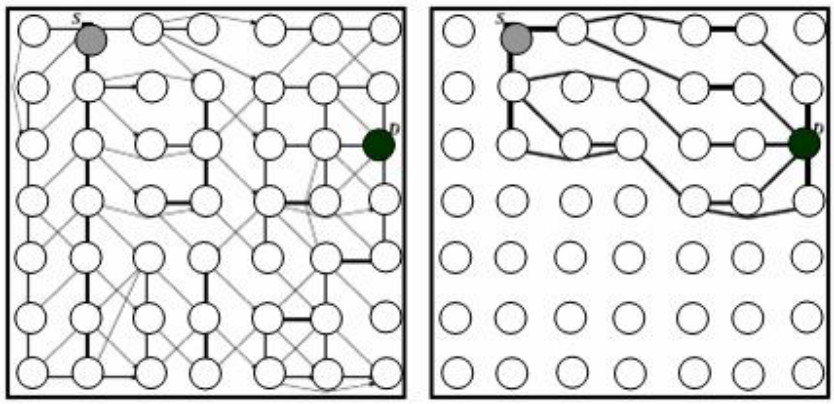


Figure 2.18: (a) RREQ Propagation (b) available Paths

Route maintenance is also similar to DSR. But since in SMR as there are more than one route available, the source node has two possibilities in case of route break. If the source node is informed that a link is broken and the session is still active, the node can therefore initiate immediately a route recovery process or use the other route and initiate a route discovery process only if all available routes to that destination are broken.

2.3.3.2. SMR Allocation Granularity

The SMR Allocation Granularity feature depicts the strength of the SMR protocol (as compared to single route protocols). If there are more than one route available to the desired destination, traffic can be distributed on these routes, for example, using per-packet allocation scheme or per stream. However, re-sequencing of packets is required upon arrival at destination, which may cause problems with TCP. But there are simple ways to solve such problem, for example, using of reordering buffer.

2.3.3.3. SMR Discussion

SMR protocol uses multiple routes per data session. The use of fastest (shortest) route minimizes the preamble time prior to sending data to a sink. Intermediate nodes also use less memory for routes since these routes are maintained by the source node.

On the other hand, SMR is not suitably to coexist with TCP, though there is a possible solution. The discovered paths may be not 100% disjoint. The discovery of multiple paths also generates more overhead on the network as compared to DSR during route discovery.

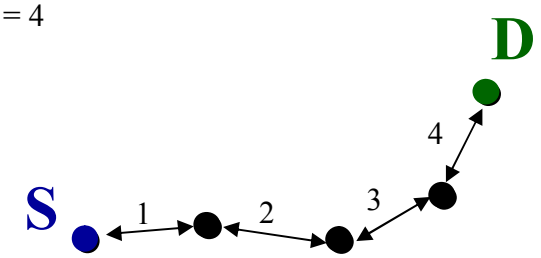
2.3.4. MP-DSR Protocol - Multipath Dynamic Source Routing [5]

MP-DSR is an extension of DSR with QoS support. MP-DSR tries to forward packets on multiple disjoint paths with certain end-to-end reliability requirements. This reliability considers the probability of having a successful transmission between the nodes, which calculated by the following equations:

$$\prod_{S,D}^{k}(t) = \prod_{(m,n)\in k} A_{m,n}(t) ----> (1)$$

path reliability

Example: k = 4

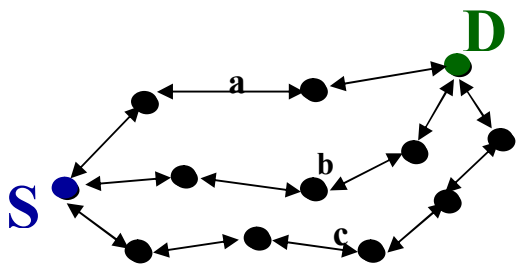


The path reliability of the k path between node(S) and node (D)

$$P(t) = 1 - \prod_{k\in K}(1 - \prod_{S,D}^{k}(t)) ----> (2)$$

End-to-end reliability

Example:



K = number of routes
K(k) = number of links in one route from node[S] to node[D]
K = 3, (a, b, c).
K(1) = 3, K(2) = 4, K(3) = 6

End-to-end reliability of K routes between node(S) and node (D)

$$\prod_{lower} = 1 - \sqrt[m0]{1 - Pu} ----> (3)$$

The lowest path reliability requirement

2.3.4.1. MP-DSR Routing

After the value of lowest path reliability requirement and the number of paths to be discovered are set, the source node floods a RREQ packet for a set of paths (neighbour nodes), which can satisfy these requirements. Each RREQ packet contains additional parameters e.g. the reliability requirement, time window, path that a RREQ message has traversed, etc. Whenever an intermediate node receives a RREQ, it checks, whether the RREQ meets the lowest path reliability requirement.
If yes, the intermediate node adds itself and sends out multiple copies of this RREQ to its neighbours. Otherwise RREQ packet is dropped. After the reception of the first RREQ, the destination waits a period of time. Then it selects multiple disjoint paths out of all received RREQ packets. RREP packets are sent along these paths.

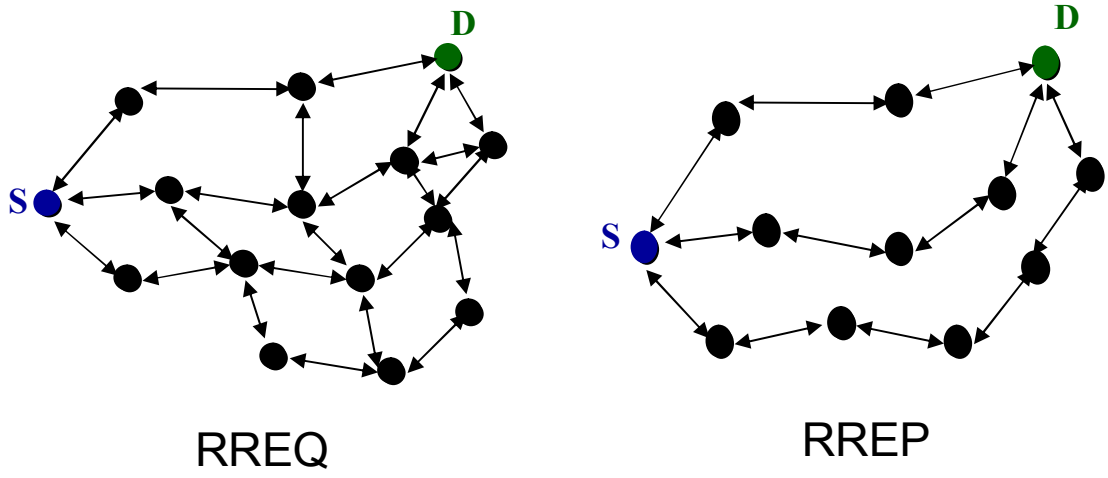
With some similarity to SMR, if a single link is broken, the source node just stops to send data through it. However, such failure does not require the route maintenance. The route maintenance takes place under two cases: the first case, when the time window at the source node expires. Here, it requires examination of up-to-date reliability. And the second case, when all paths are broken at the time instant. The source node initiates directly a new route discovery. In addition to the maintenance of each individual route MP-DSR protocol performs an end-to-end reliability repair.

2.3.4.2. Discussion

MP-DSR protocol supports end-2-end reliability based on the link availability and path reliability model. It builds multiple disjoint paths. This means that the data transmission fails if and only if all disjoint paths fail at the same time. On the other hand, it needs time till the first RREP comes back. Therefore, the nodes require more buffer space or high packet loss is results. MP-DSR uses periodic message exchange which leads to more overhead.

Some possible improvements are the reply on shortest delay path, which in most case could be the best/shortest path, and use of overhearing, instead of periodic beaconing.

2.3.3.3 Example



100% multiple disjoint paths compared to SMR, see SMR scenario on page 40

Figure 2.19: MP-DSR RREQ & RREP messages

3. PROTOCOL IMPLEMENTATION

3.1. Implemented Protocols

During the thesis the routing protocols DSR and SMR have been implemented with the following features

3.1.1 DSR

The two main mechanisms of DSR route discovery and route maintenance are completely implemented. Moreover, the some additional features of the route discovery are also implemented.

- Route Discovery
  - Route Request
  - Route Reply
  - Caching Overheard Routing Information,
  - Replying to Route Requests Using Cached,
  - Route Request Hop Limits.
- Route Maintenance
  - RERR

In order to implement the above mentioned features the following data structures have to be added:

- Route Cache Table
- Send Buffer
- Route Request Table
- Blacklist

3.1.2 SMR

As mentioned before, SMR protocol is based on DSR protocol. In addition to DSR all three new features mentioned in SMR paper [4] are implemented:

- Forward of duplicated RREQs with respect to routing regulation.
- Features of the shortest path, wait period, temporary route cache table and build of max disjoint paths.
- Using of multiple routes per one data session. The “per packet” schema to split the data traffic has been used.

3.2. Mobile Host Protocol Structure

The simulations written during this thesis are based on OMNeT++ 3.3 under Linux (ubuntu 7.10) operating system; see Section 4.1 “OMNeT++ Overview”. The protocols have been implemented as an adaptation of the INET framework. Extensions of the network and MAC layers of INET are required to support source routing, see Section 4.2 “Integration with INET”.

Each mobile host is a compound module *(Figure 3.1)*, which encapsulate the following simple and compound modules:

1. Application Layer – chooses the destination and generates the data packets. It sends its data to the transport layer. We used UDPBasicApp and UDPVideoStream applications.
2. Transport Layer – encapsulates/decapsulates the packet arriving from higher/lower layers and sends it to the next layer. We used UDP Transport application.
3. Route Module – where the routing mechanisms are implemented. It sends its routing packets down to the transport layer.
4. Routing Data Queue – queues data packets comes from application layer which has not yet an available route.
5. Network Layer – the main simple module is the IP. It communicates at most with the routing table.
6. Routing table – saves records of route to destinations.
7. Interface table – saves records of interfaces. For example: wlan.
8. MAC Layer - identifies each node in the network and sends ACK, CTS/RTS packets.
9. Physical Layer - makes link to other host inside its own transmitting range, makes the actual transmitting and checks packet's bit error.
10. WLAN Network card – is a compound module of MAC and Physic layers.
11. Mobility module – is considerate of the movement of the hosts.
12. Notification Board Module – is a notify system which can be used/called in all layers.

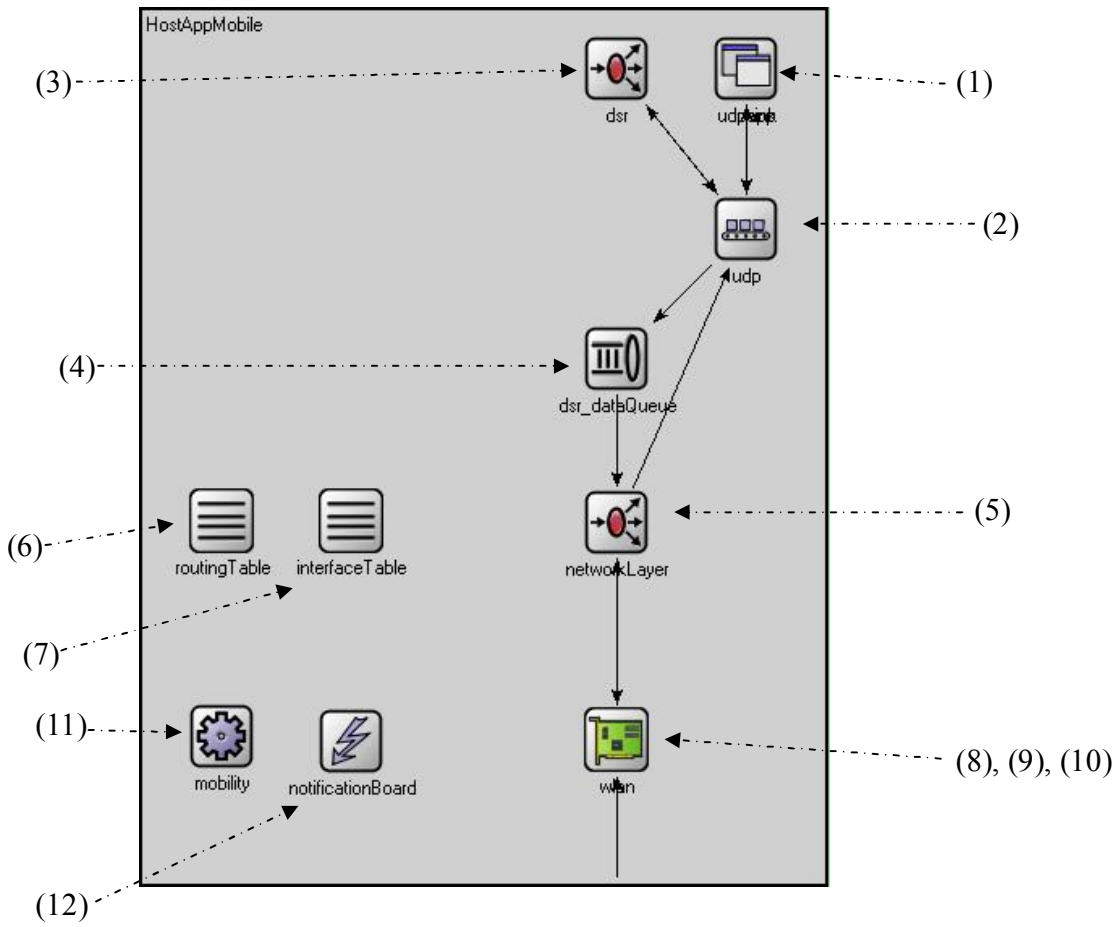


Figure 3.1 Host layer structure

3.3. DSR UML Diagrams

3.3.1. DSR Class Diagram

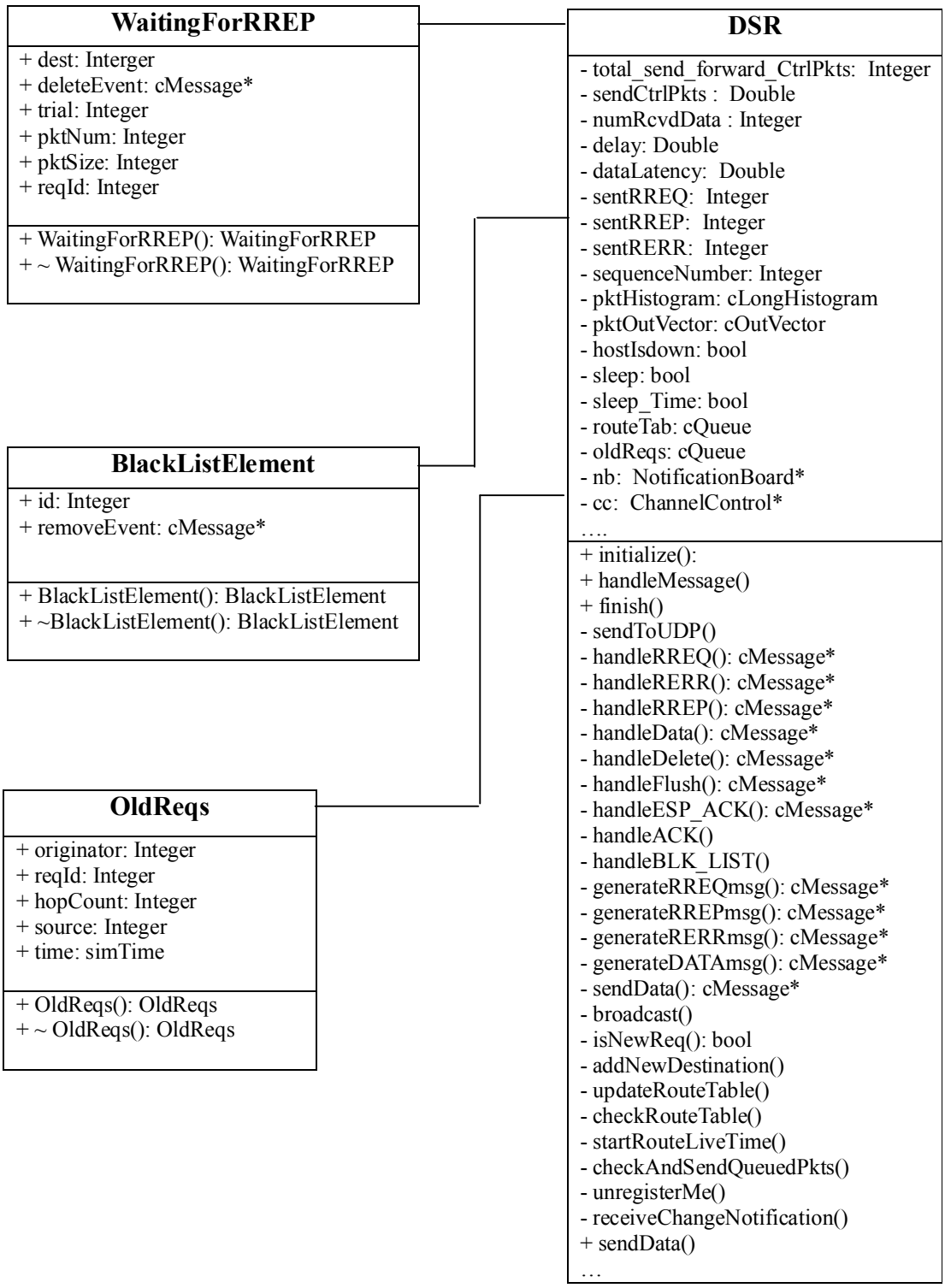


Figure 3.2 DSR class diagram

3.3.2. DSR Route Discovery “RREQ & RREP”

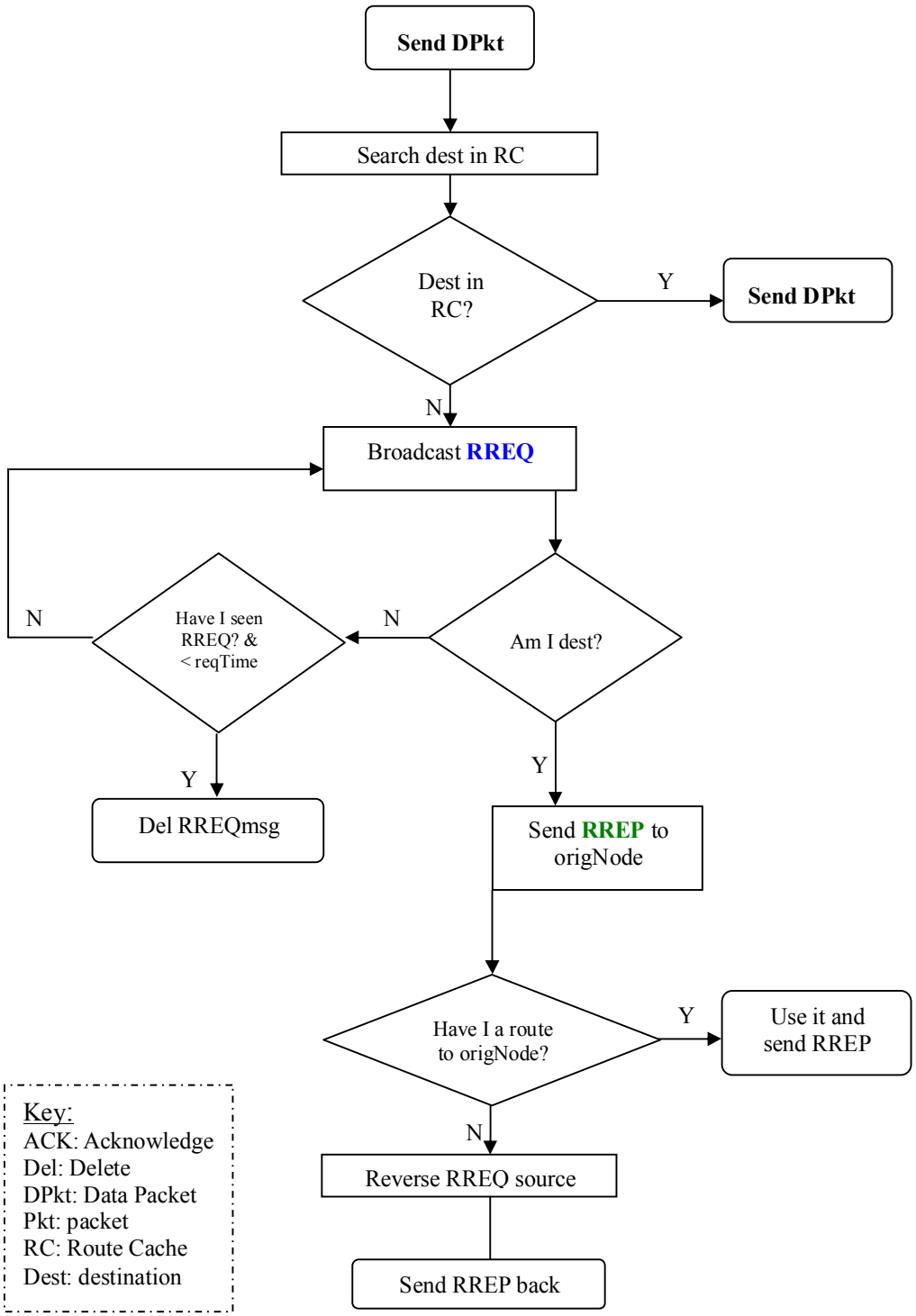


Figure 3.3 DSR route discovery “RREQ & RREP”

3.3.3. DSR Transmission of a Data Packet:

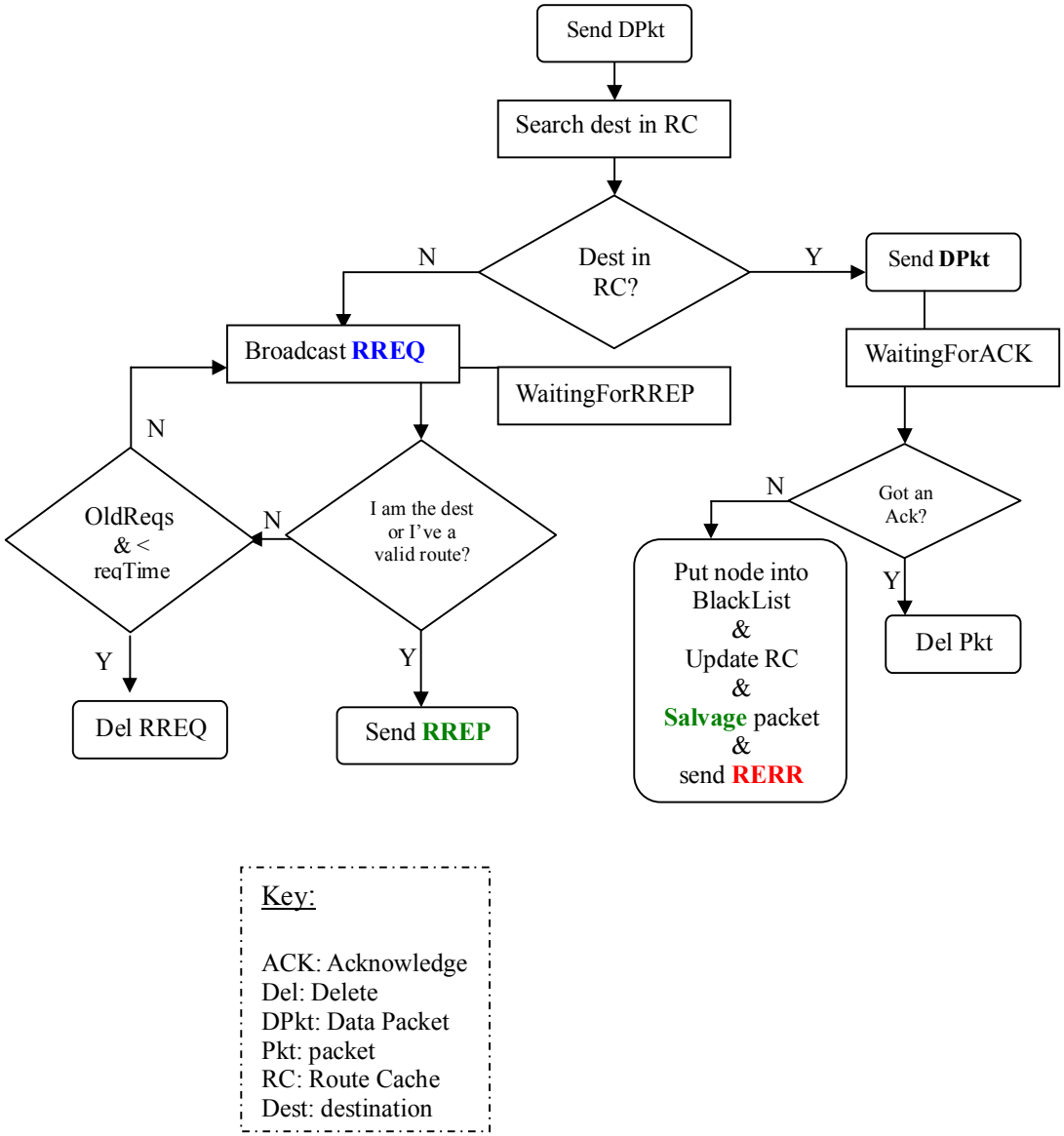


Figure 3.4 Send Data Packet using DSR protocol

3.3.4. DSR Route Maintenance

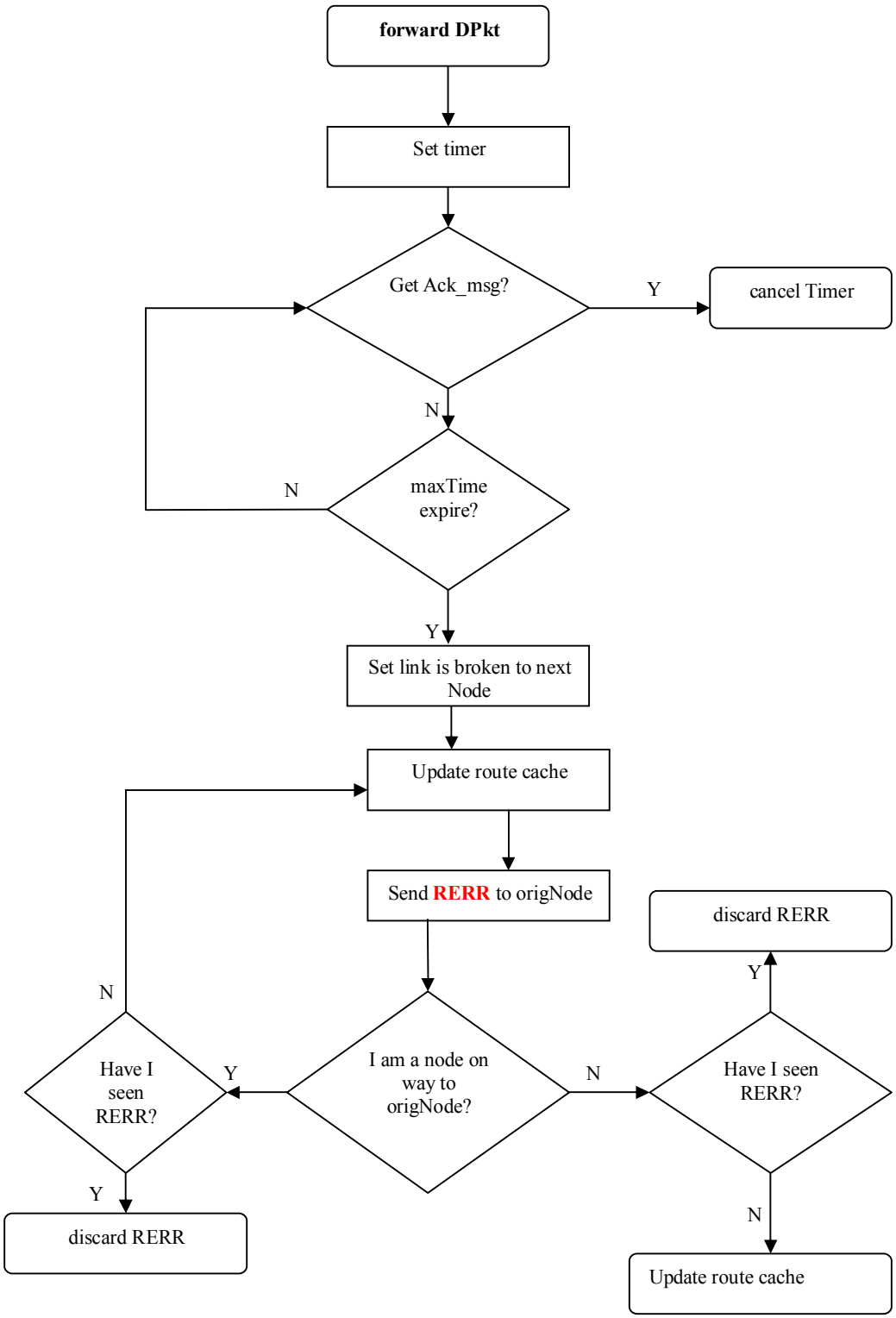


Figure 3.5 Basic DSR Route Maintenance

3.3.5. DSR Test Scenarios

We used the scenarios, a), b), c) and d), to validate our implementation. The DSR protocol uses the a) RREQ message to send request and uses the b) RREP message to reply, then when it has a valid route c) sends data to desired destination. In case of an error accurse d) RERR message will be sent.

The arrows in a), b) and d) describe the sending of routing packets and their complexity related to the overhead on the network.

a) RREQ: Node [A] propagates a route request to Node [G]

Node [A] wants to send data to destination node [G], but it has no route yet! So it starts a route request discovery. Additionally, we assume that all intermediate nodes have no valid route - in their route cache table - to destination node [G]. Blue arrow represents the actual broadcast of RREQ packet, black arrow represents the old process of RREQ broadcast.

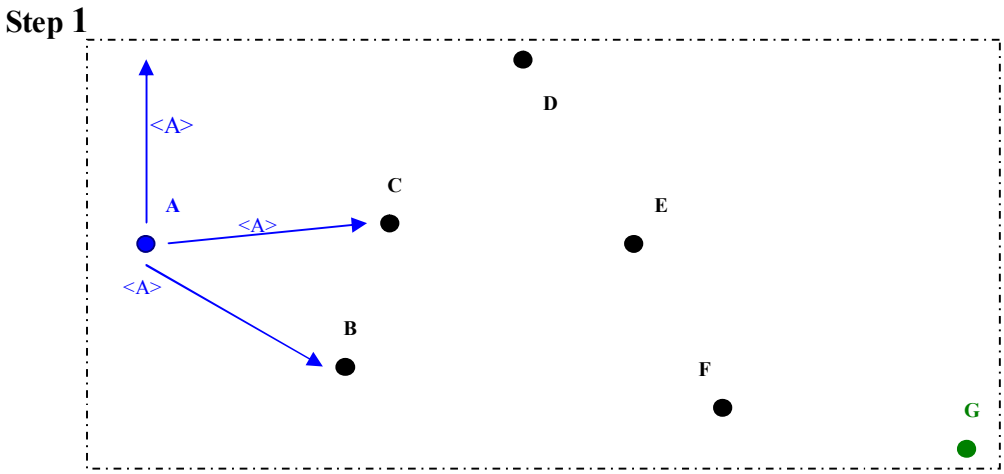


Figure 3.6.1 DSR RREQ propagation

In step 1, node [A] broadcasts a RREQ packet (blue arrow) to destination node [G] and appends its own address in the route record field on the packet header. The RREQ packet is received by all nodes within the transmission range of the initiator node (node [A]). The RREQ packets arrive at node[C] and node [B].

Step 2

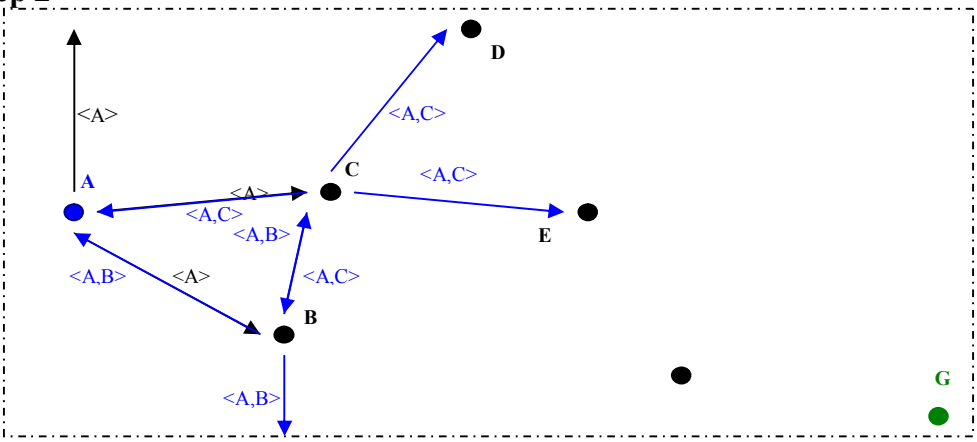


Figure 3.6.1 DSR RREQ propagation

In step 2, node [C] and node [B] rebroadcast the RREQ packet and appends their own addresses in the route record field. RREQ packet is received by all nodes within the transmission range of node [B] and node [C]. The RREQ packets from node [B] arrive at nodes [A and C] and from node [C] is received by nodes [A, B, D and E]. Node [A] ignores these RREQ packets, as it is the initiator of the packet respectively it finds its own address in the route record field. Node [B] and node [C] ignore the RREQ packets from each other as well. These RREQs have been already processed (i.e. old RREQ). Nodes [D and E] received the RREQ packet from node [C].

Step 3

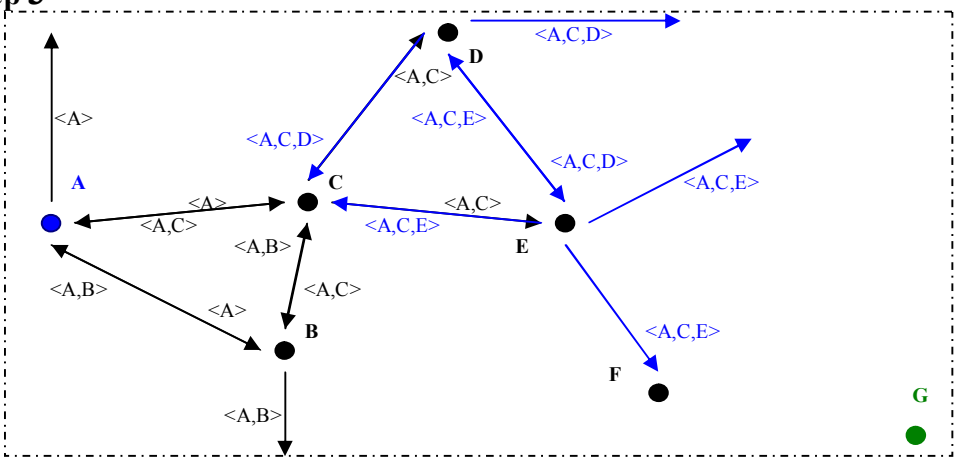


Figure 3.6.1 DSR RREQ propagation

The step 3 works in the same way as step 2. The RREQ packet is rebroadcast by node [D] and node [E] after they have appended their own address in the route record field. Duplicate and old RREQ packets are ignored by nodes [C, D and E]. Node [F] receives the RREQ packet from node [E].

Step 4

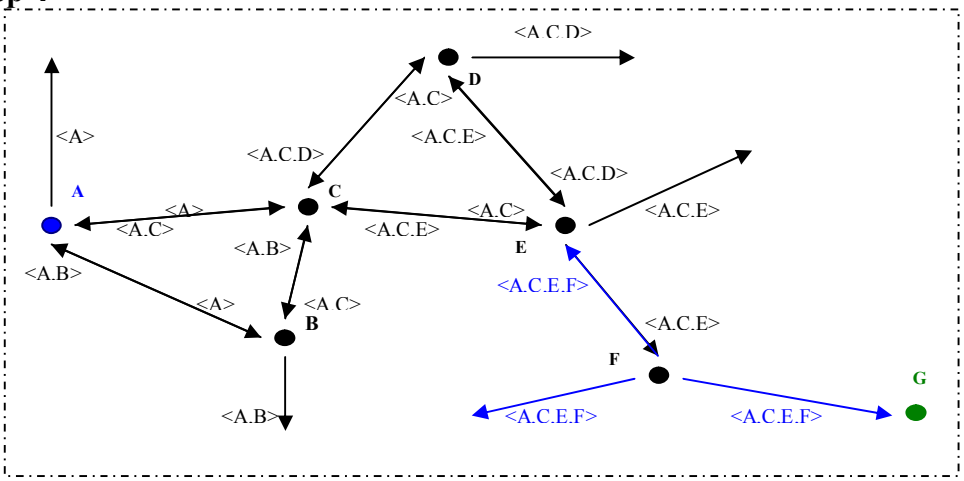


Figure 3.6.1 DSR RREQ propagation

In step 4, node [F] appends its own address in the route record field and rebroadcasts the RREQ packet. Node [E] ignored this RREQ packet from node [F] as it is an old one. Node [G] receives the RREQ packet from node [F].

Now, the RREQ packet arrives at the destination node [G]. Then node [G] replies with a RREP packet to the initiator node [A]. The RREQ packet arrives its destination via nodes [C, E and F]

b) RREP: Node [G] sends a route reply to Node [A]

We assume that destination node [G] has no other route in its route cache table. It sends the RREP packet via the reverse path as an unicast packet.

Step 1

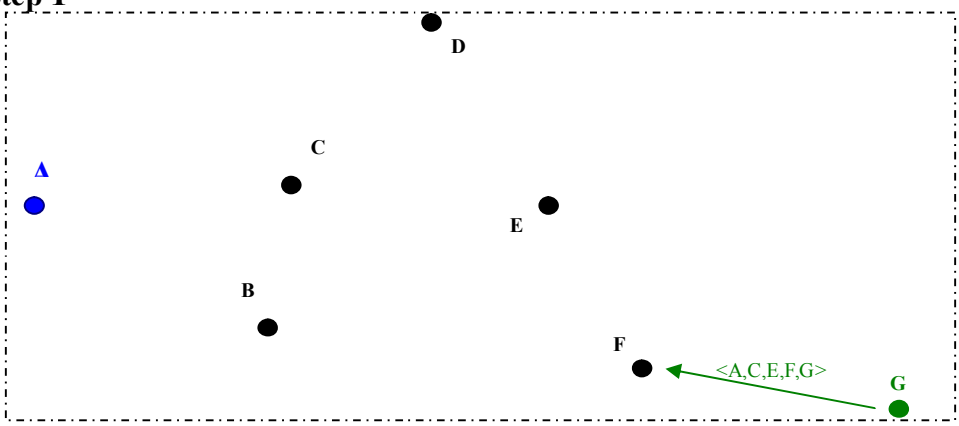


Figure 3.6.1 DSR RREP

In step 1, the node [G] sends the RREP packet to the node [F]. RREP packet includes the traversed path by RREQ packets.

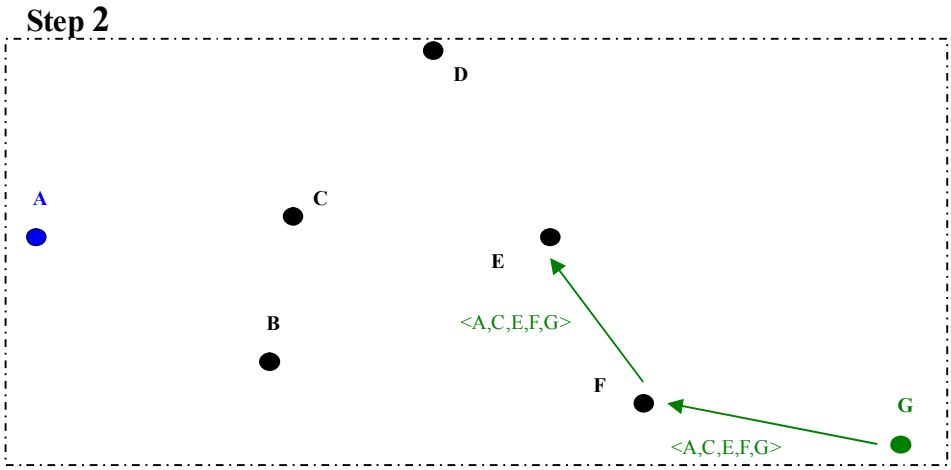


Figure 3.6.1 DSR RREP

In step 2, node [F] checks if it the destination for this RREP packet. Otherwise, it forwards the RREP to the next hop on the source route field (i.e. node [E]).

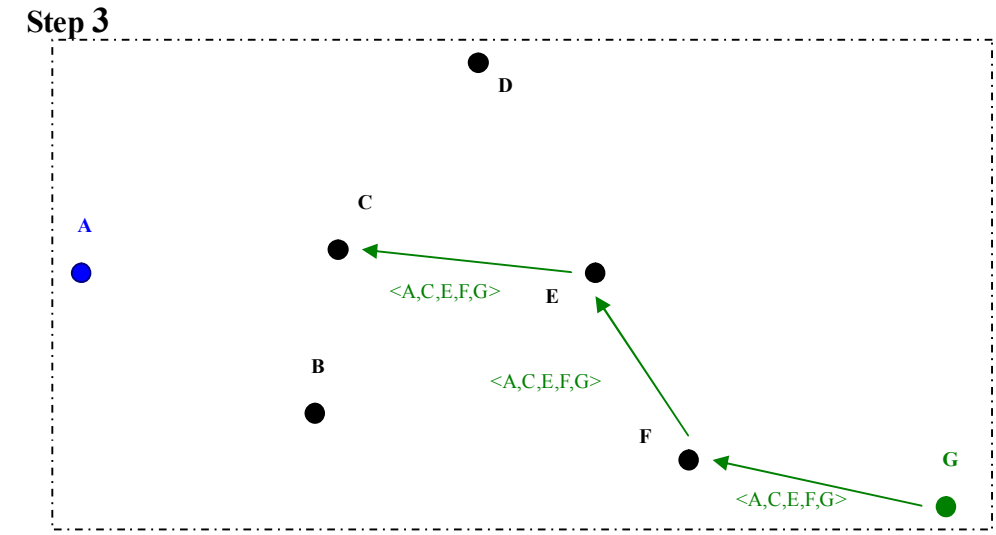


Figure 3.6.1 DSR RREP

In step 3, it is similar to step 2. Node [E] forwards the packet to node [C].

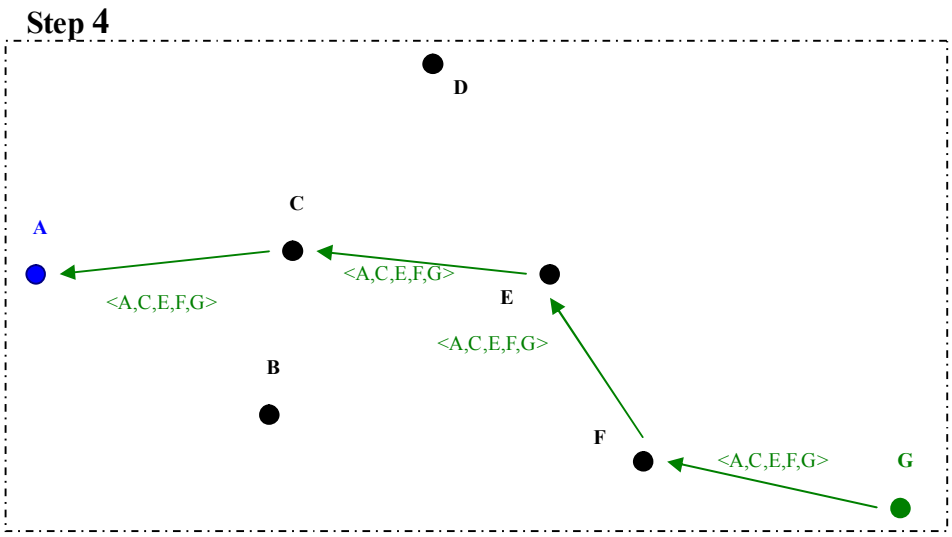


Figure 3.6.1 DSR RREP

In step 4, the packets are processed as in step 2. Node [C] forwards the RREP packet to next node [A]. Whenever a node receives a RREQ or RREP packet, it adds all usable routing information from the packet in its route cache table.

Now, node [A] has a valid route to node [G] via: A → C → E → F → G, and can start to transmit its data packets.

c) Send data: node[A] sends data to node[G]

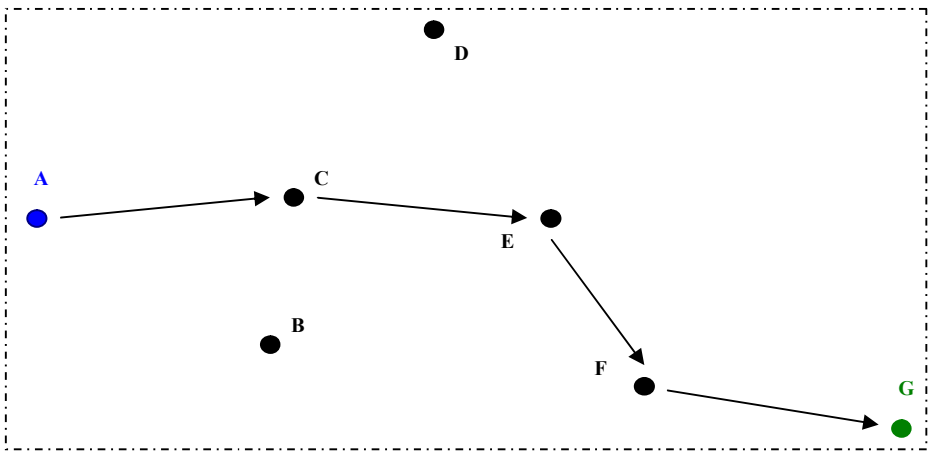


Figure 3.6.1 DSR send data

Whenever a node receives a data packet, it acknowledges the reception by sending an ACK to the source (previous) node.

We assume that “at least” one of the nodes over this discovered path has moved out of the transmitting range of its previous node, during an active session. Now, we have a broken link between these two nodes.

In such case, the previous node will send back a RERR packet to the source (initiator) node as unicast packet.

d) RERR: link between node [E] and node [F] is broken

Step 1

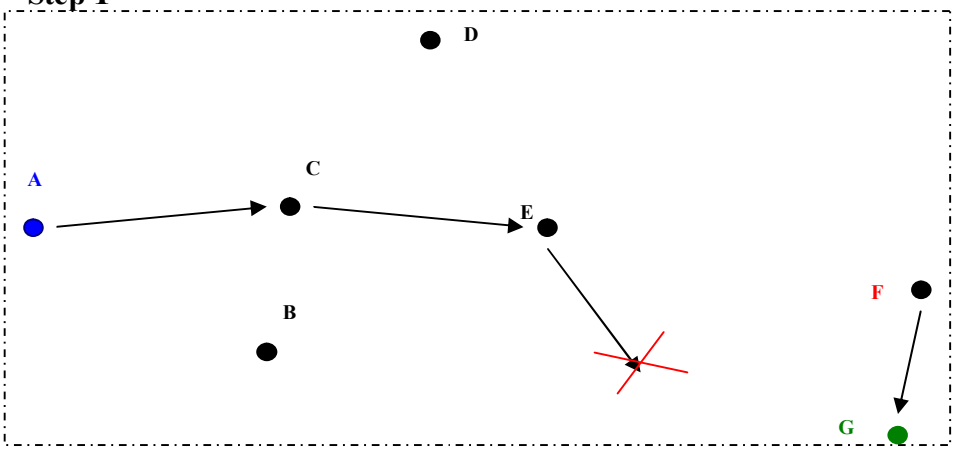


Figure 3.6.1 DSR link broken

In step 1, node [F] moves out of the transition range of node [E]. So, if node [E] forwards a data packet to node [F], it does not receive any ACK message. After some tries, it considers this link as a broken link and node [F] no more reachable through this path.

Step 2

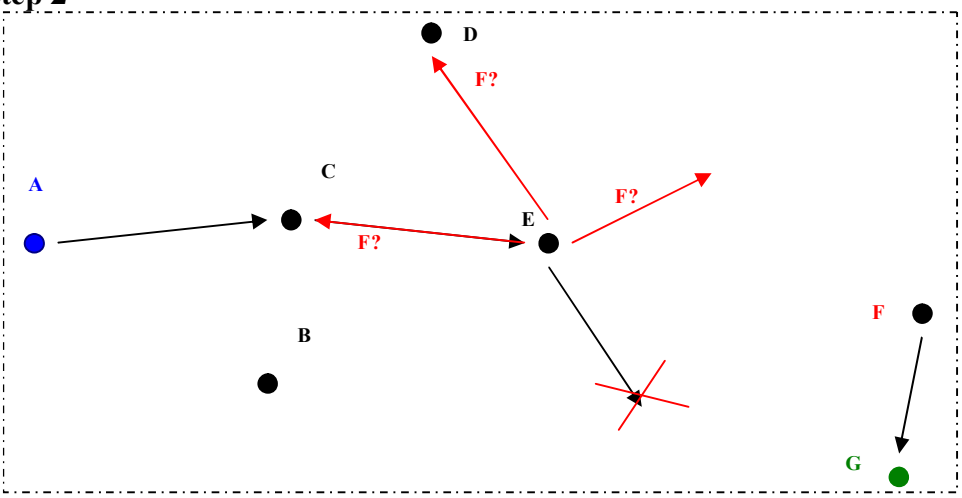


Figure 3.6.1 DSR RERR

In step 2, node [E] generates an RERR, puts the previous node [C] as next hop and sends it back to the source node [A] as an unicast packet.

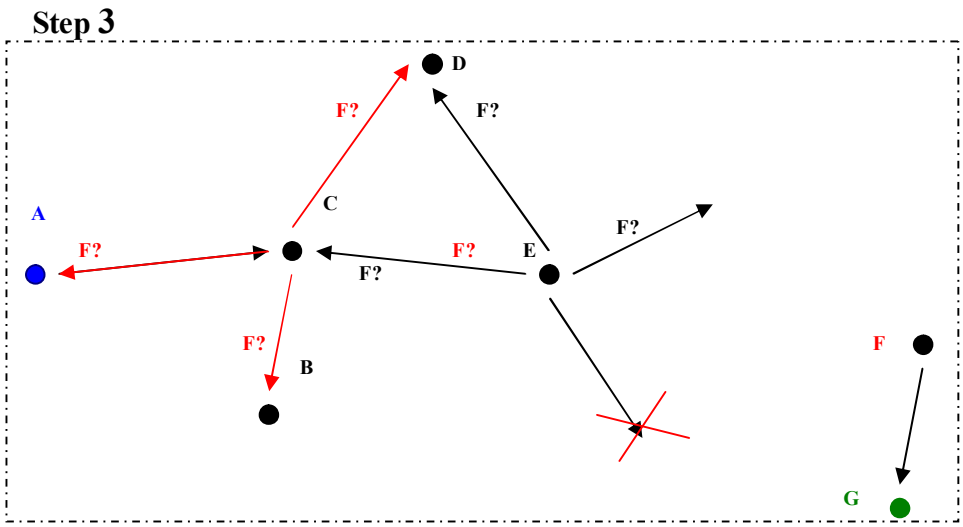


Figure 3.6.1 DSR RERR

In step 3, node [C] forwards the RERR to node [A]. Then, node [A] updates its route table, and if the session is still active and if there is another valid route to destination in its route table, node [A] uses this route, otherwise it starts a new route discovery process. If the nodes [D and B] receive the RERR packet, they update their route cache table, but they do not forward the packet, insofar it does not address them.

Whenever a node receives an RERR packet, it deletes all route entries in its route cache table, which includes this broken link.

3.4. SMR UML Diagrams

3.4.1. SMR Class Diagram

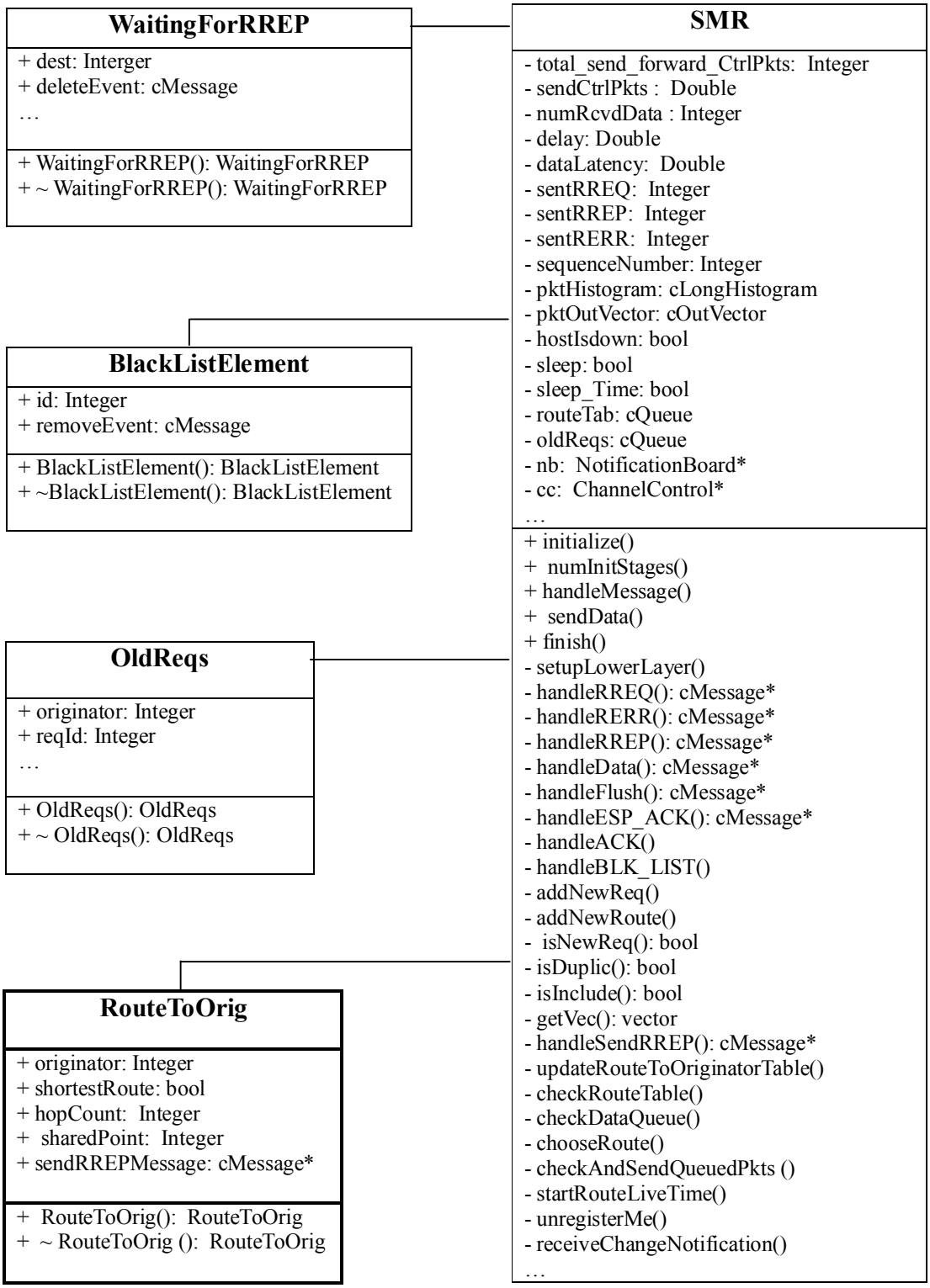


Figure 3.7 SMR class diagram

3.4.2. SMR Route Discovery: RREQ & RREP

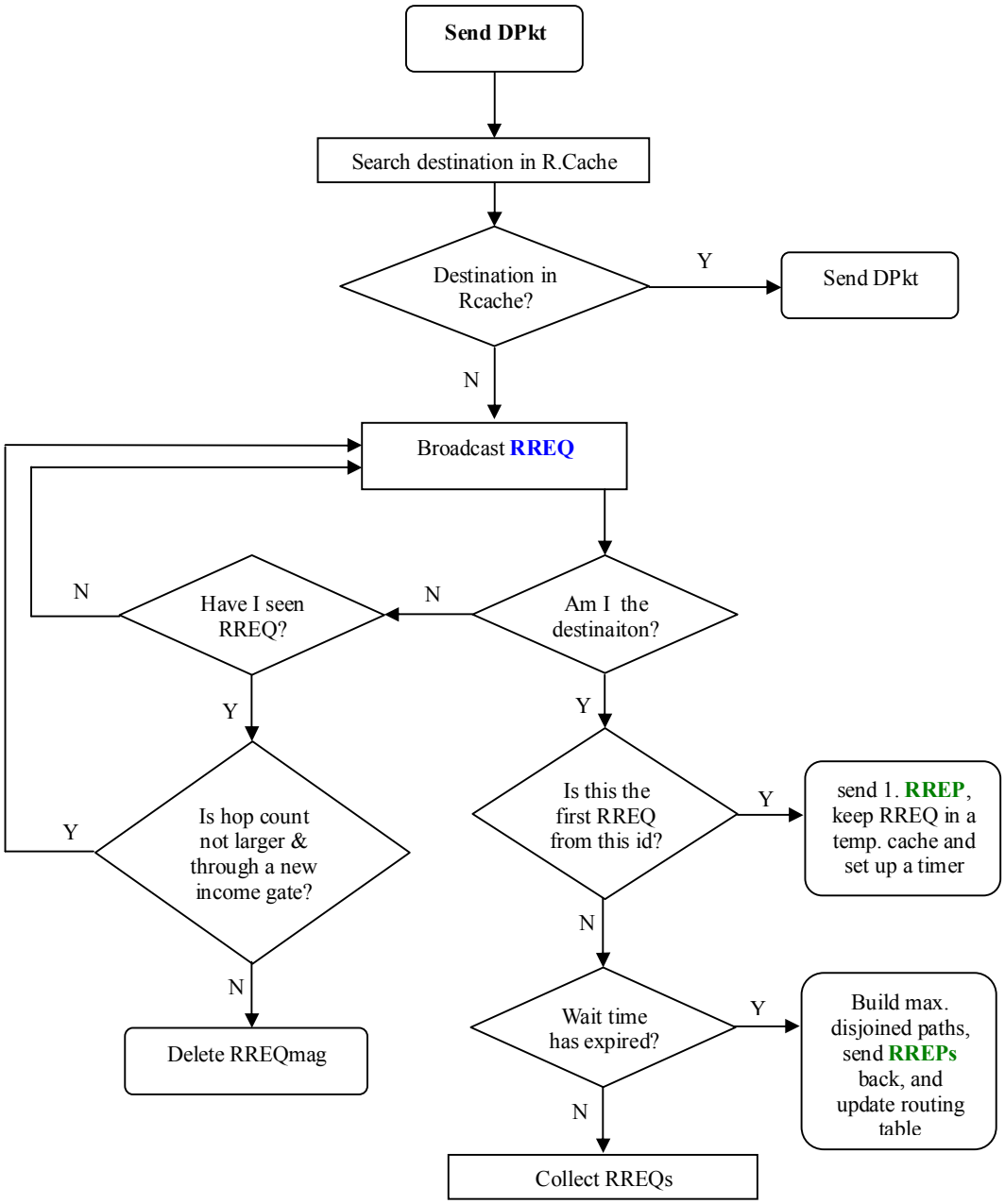


Figure 3.8 SMR route discovery "RREQ & RREP"

The route maintenance flow chart for SMR is same as for DSR on p30. Therefore, it is omitted.

3.5.3. SMR Test Scenario: RREQ & RREP

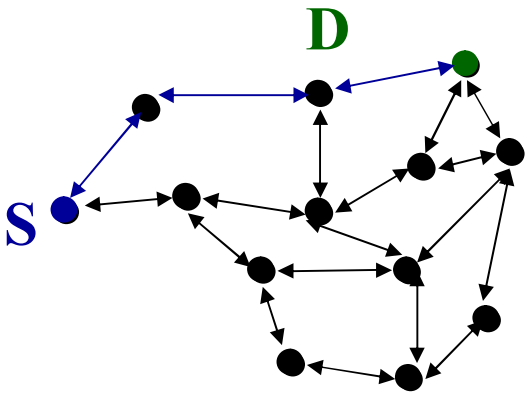


Figure 3.9.1 SMR RREQ

The blue path is the shortest delay path. And it is the first route path (1) to be selected. Then, the node [D] selects after a period of time max. disjoint paths (2 and 3) and sends RREP back to node[S] via them.

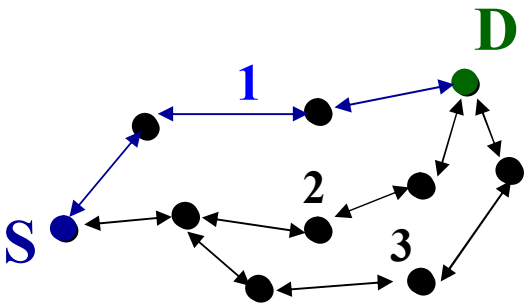


Figure 3.9.2 SMR RREP

4. SIMULATION

4.1 OMNeT++ Overview [10]

OMNeT++ is a public-source, component-based, modular and open-architecture simulation environment. It developed by Andras Varga at the Technical University of Budapest. Its primary application area is the simulation of communication networks and it has been successfully used in other areas like the simulation of IT systems, queuing networks, hardware architectures and business, because of its generic and flexible architecture. Most amazing feature by OMNeT++ is its graphic user interface - GUI *(Figure4.1)*. It has a very good, easy and helpful GUI tools especially while debugging. The second thing is the documentation (user manual and examples) which is well formed.

There are some frameworks that could be integrated with OMNeT++ (e.g. INET Framework, Mobility Framework). The INET Framework supports wired, wireless and mobile simulations. Support for mobility and wireless communication has been derived from the Mobility Framework. There are several protocols and applications are implemented IPv4, IPv6, TCP, UDP. The framework includes also 802.11, Ethernet, PPP, RIP, OSPF, MPLS with LDP and RSVP-TE signalling, NesCT and other protocols. But it still not covers the ad hoc network areas.

OMNet++ uses two types of modules: simple modules and compound modules.

- Simple module is a C++ class inherited from cSimpleModule. It is identified by name. And it is defined by declaring its parameters and gates. The Simple modules are the basic submodules for other (compound) modules. A simple module is defined by declaring its parameters and gates. [20]
- "Compound modules are modules composed of one or more submodules. Any module type (simple or compound module) can be used as a submodule. Like simple modules, compound modules can also have gates and parameters, and they can be used wherever simple modules can be used."[20]

To create a simple module in OMNet++, it (C++ class) has to be registered with OMNeT++ via the Define_Module() macro and the following three functions have to be overwritten:

- initialize()
- handleMessage(cMessage *msg)
- finish()

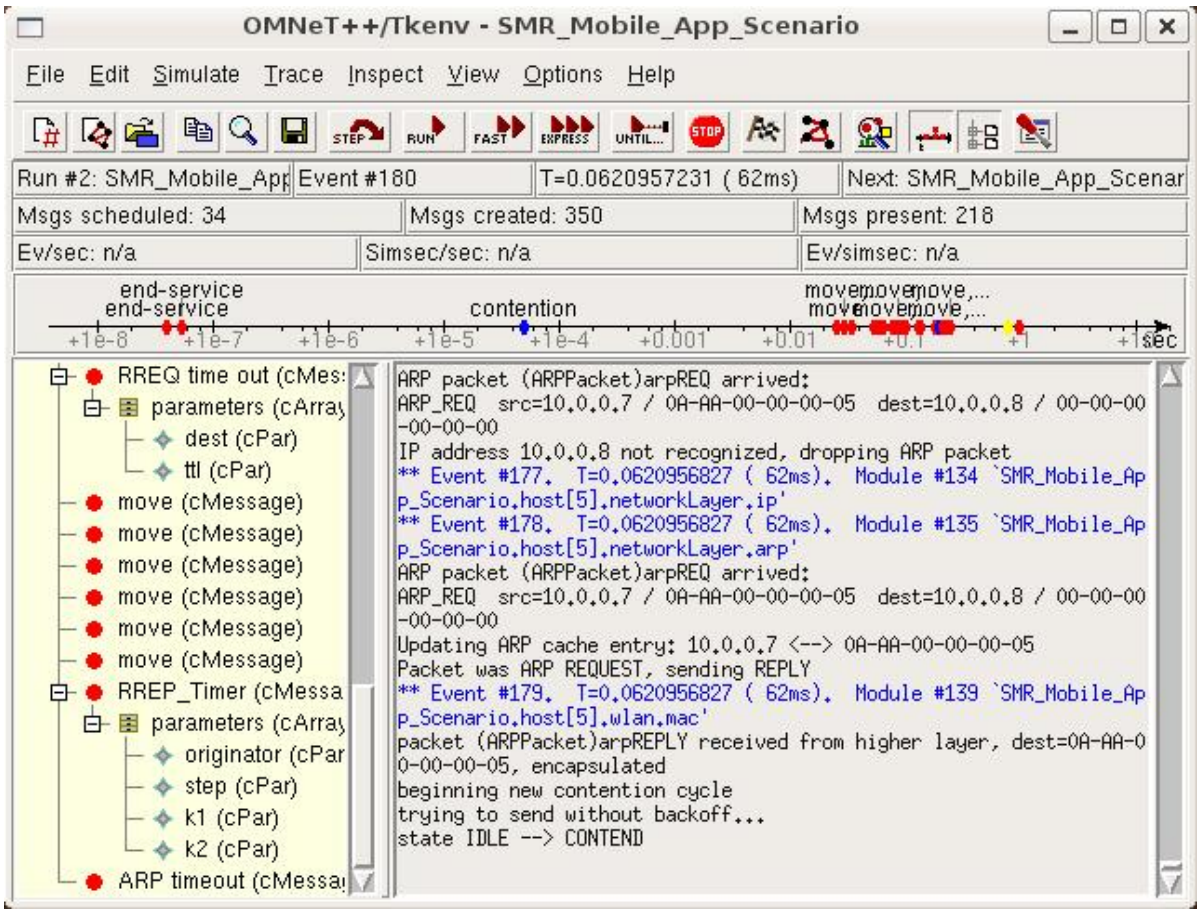


Figure 4.1 OMNeT++ Main window

4.2 Integration with INET Framework

4.2.1 Configuration Files

In order to get our own modules work with OMNeT++ simulator and INET framework, two types of configuration files has to be created, namely NED and INI files. A NED file defines the simple and compound modules. It creates the gates and establishes the connections between these modules. This file builds the actual network infrastructure. An INI file has many sections. In every section there are particular setting has to be defined. For example: the setting of the parameters which in NED files are defined, in the INI file.

- My NED Files

  For each scenario category there are two NED file that have been created, namely a host and a scenario NED files. A host file declares the whole layers structure, which is used in this implementation (the Application, Transport, Network, and Physic layers). A scenario file defines the actual network, *example 4.2*.

- My INI File

  It defines the values for variables which are declared in NED files, and runs the scenarios, *example 4.3*.

```
import
   "DSR",
   "RoutingTable",
   "NetworkLayer",
   "UDP",
   ……..

// Wireless-enabled Host running DSR. It builds node's structure
module HostMobileApp
   gates:
      in: radioIn;

   submodules:
         notificationBoard: NotificationBoard;
            display: "p=140,462;i=block/control";
         dsr: DSR
            display: "p=304,47;i=block/fork;q=queue";
         dsr_dataQueue: DSR_DataQueue;
            gatesizes:
                  toNetwork[1],
                  fromTransport[1];
            display: "p=304,226;i=block/queue";
         udp: UDP;
            display: "p=384,146;i=block/transport";
         ……

   connections nocheck:
      udp.to_app++ --> dsr.from_udp;
      udp.from_app++ <-- dsr.to_udp;

      dsr_dataQueue.fromTransport[0] <-- udp.to_ip;
      ......
endmodule
```

Example 4.2: NED file

```
-[General]
preload-ned-files = *.ned @/home/abdalla/INET/nedfiles.lst

network = scenario
sim-time-limit = 900h

[Cmdenv]
express-mode = yes

[Tkenv]
default-run = 1

[Parameters]
*.playgroundSizeX = 250 ;[m]
*.playgroundSizeY = 250 ;[m]

; Hosts
*.hosts = 5

; DSR
*.host[*].dsr.DSR_INTERFACES = "wlan"

[OutVectors]
**.udpapp.*.enabled = yes
```

Example 4.3: an omnetpp.ini file

4.2.2. Example: SMR Mobile Host with INET Framework

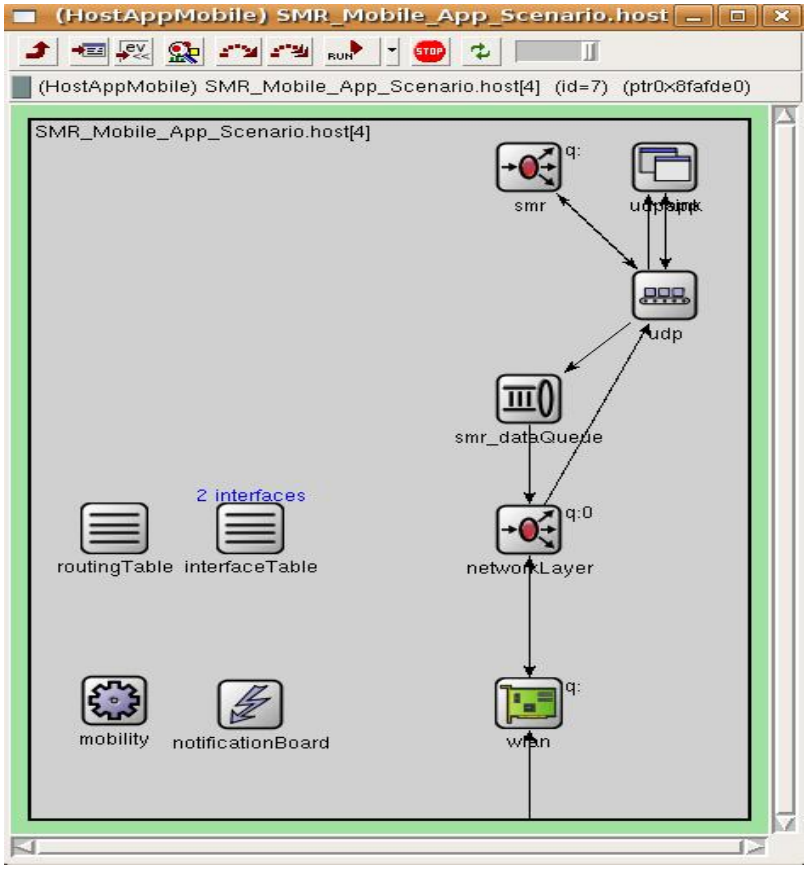


Figure 4.4 SMR integrated with INET framework

4.2.3. INET Problems

During the integration of the new modules in the INET framework, several problems arise. The found solutions are described in the following.

- First Problem

  Network layer of INET framework does not support the source routing! Some parameters are defined; however, the routing mechanism of the framework works as hop-by-hop routing. There are no functions to add the whole path vector to the packets header. Whenever a data packet arrives at the network layer of an intermediate node, the network layer searches its routing table for the next hop toward the destination.

  Solution

  We implement the required functions to add the source route vector to data packet header. Furthermore, the routing functions have to deal with some more cases. *Figure 4.5* describes these cases. For more details see the IP.cc file of the INET framework.

```
if(packet comes from Higher Layer) {
      if(nodeSupportSourceRouting()){
            // use IP routing (lookup in routing table) for destAddr
            if(packet destAddr is broadcast) {// (case of RREQ)
             then fragment and  forward the packet
                return;
            }
        if (no entry for the destAddr is found)
         {
        // error handling: destination address does not exist in
         }
        if(packet's source Routing Option is Unspecified)
        {
        //add Source Route to Packet's header;
        }
      }  else // node does not  support source route
      {
        // do as it was already! Hop-by-hop forwarding
      }
 }  else // packet does not come from higher layer
 {
      if(packet's source Routing Option == 4){// 4 is the source routing option value
          if (destAddr is broadcast)
          { then forwards to higher layer (case of RREQ)
            return;
          }
        else // Forwarding
        {
          // get next hop from packet's header " source route" and forward
         return;
        }
    } else // packet is not source routing option
    {
      // handle as it was alredy!
    }
}
```

Figure 4.5 Pseudo code of the IP routing algorithm

- Second Problem

The structure of the **Routing Entry** of INET framework has to be modified to support source routing algorithm.

Solution

The RoutingEntry class has been extended and some new functions and parameters were added to the RoutingTable class to support source routing algorithm. *Example 4.6* shows the new routing file structure.

```
route:
IP Address   Gateway    Net mask          source vector                  direct/remot route   Interface
10.0.0.20    10.0.0.13  255.255.255.255   10.0.0.13 10.0.0.7 10.0.0.20   remote               wlan
10.0.0.40    10.0.0.33  255.255.255.255   10.0.0.33  10.0.0.40           direct               wlan

routeend.
```

Example 4.6 Routing file

- Third Problem

  If a data packet could not be delivered to the next hop, the MAC Layer does not return any kind of information to the upper layers. In addition, some times the MAC queue stays in undefined/endless state.

Solution

Cross layer information is set to the different layers by porting them to the notification board of INET. For the state problem, the responsible function has to deal with some more cases. For more details see the “beginNewCycle” function in Mac80211 class of INET. But it requires more developing and synchronizing with other states of the physic layer.

5. PERFOMANCE EVALUATION

5.1. Simulation Scenarios

We evaluate the following protocols:

- DSR-1: DSR with intermediate node routes replying on RREQs
- DSR-2: DSR without intermediate node routes replying on RREQs
- SMR: Split Multipath Routing uses multiple routes per data session and performs the route recovery only when both initially searched routes to the sink are broken.

We implemented the two protocols (DSR and SMR) in C++ language and simulated them within OMNeT++ Simulator and INET Framework. Our simulation contains two kinds of scenarios: a mobile scenario and mesh scenario. These scenarios are tested by two types of applications, UDPApp and UDPVideoStream.

- The UDPApp is an application sends data as individual packet. And it does not require any kind of registration between the sources and sinks nodes.
- The UDPVideoStream is a streaming application that uses client-server architecture. In order to get the stream, the client first sends a request to the server. After this registration. the server starts with the transmission of the video stream. As the path between client and server is already established during registration, there is no delay as well as no dropped packets at the start of the transmission.

The simulation model has the following parameters:

General parameters:

parameter	**example value**
Simulation time	600s
Number of host	25
Routing_INTERFACES	wlan
AUTOASSIGN_ADDRESS_BASE	10.0.0.0
Simulation area	1000m x 1000m , 1500m x 1500m
Mobility model	Random way-point, NullMobility
Speed	uniform(0, 10)m/s
Pause time	0s, 50s, 100s, 300s, 600s
Host failure precent	0%, 5%, 10%, 20%, 30%
Transmission range	250 m
Channel capacity	11 Mb/s
Payload size	1450 byte
Packet rate	uniform(0.0604, 0.0608), uniform(0.0202, 0.0204)
Traffic type	UDPApp, UDPVideoStream
Failured host (sleep)	true
Failure_Time_Start (sleep_Time)	180 sec
BUFFER_SIZE_PACKETS	50
BUFFER_SIZE_BYTES	75000 byte

DSR model parameters:

parameter	example value
intermidiateNode_REPLY	false

SMR model parameters:

parameter	example value
use_roundRouting	true
use_chaching_D	true
use_chaching_R	false

5.2. Performance Results

Each scenario has been run 20 times with different seeds. There are two data sessions per scenario. We evaluated and compared the following metrics:

- **Packet Delivery Ratio** is obtained by dividing the number of the received data packets by the destinations (at the application layer) by the number of data packets originated by the sources.
- **Packet loss** is the number of dropped packets caused by unreachable destinations, network collision or bit errors.
- **Data Latency** describes the mean time (in seconds) taken by the data packets to reach their destinations.
- **Normalized Routing Overhead** is obtained by the number of control packets dividing by the number of data packets received by destinations.
- **Route Live Time** describes how long a source node has at least one route available to its destinations.

In the next part, we discuss the mobile scenarios.

5.2.1. Mobile Application Scenario (mobile app)

The mobile app scenario (Figure 5.1) models a network of 25 mobile hosts placed randomly within an area of 1000 x 1000 meters. Each host has a transmission range of 250 meters and a channel capacity of 11MB/s. Each run executed for 600 seconds of simulation time. There are two data sessions, for which the source and destination nodes are set in the configuration file (omnetpp.ini). We use the built-in UDP application (UDPApp) of the INET framework to generate traffics with uniformly distributed packet rates in the interval [0.0604, 0.0608] for low rate scenario and in the interval [0.0202, 0.0204] for the high rate scenario. The payload size is 1450 bytes. We use use the IEEE 802.11 implementation of INET (MF80211) as the medium access control protocol. Random way-point mobility is selected as mobility model for the individual nodes. Several different scenarios with different mobility degrees are tested. The mobility is adapted by selecting different pause times for the nodes in the mobility model. The travel speeds of the nodes are uniformly distributed between the constant minimum and maximum speeds of 0m/s and 10m/s.

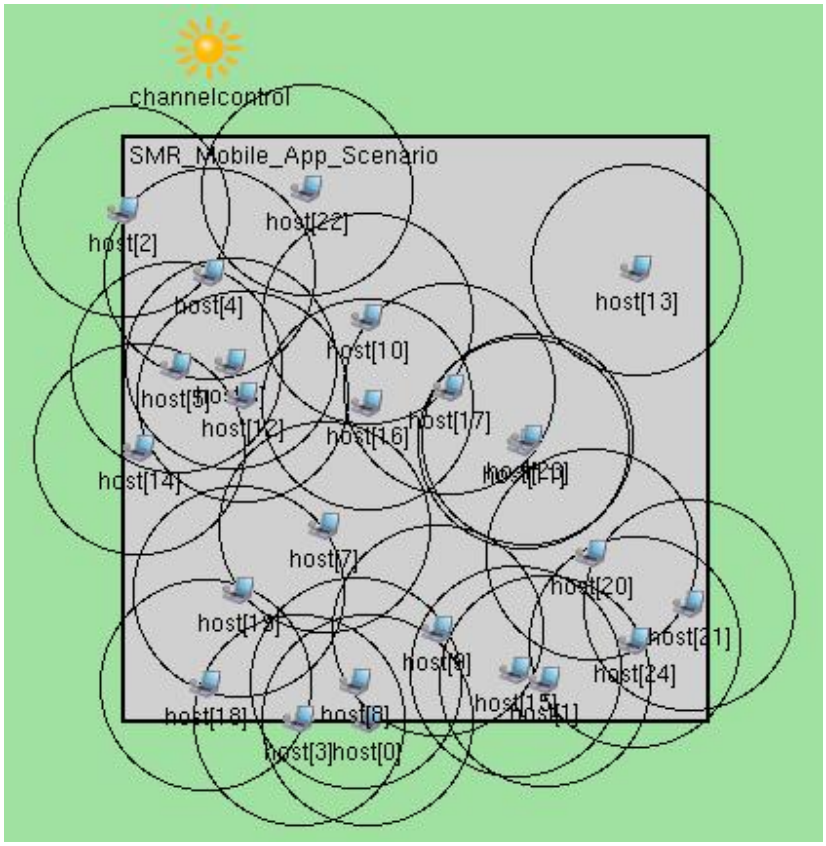


Figure 5.1 Mobile Application Scenario

5.2.1.1. Packet Delivery Ratio

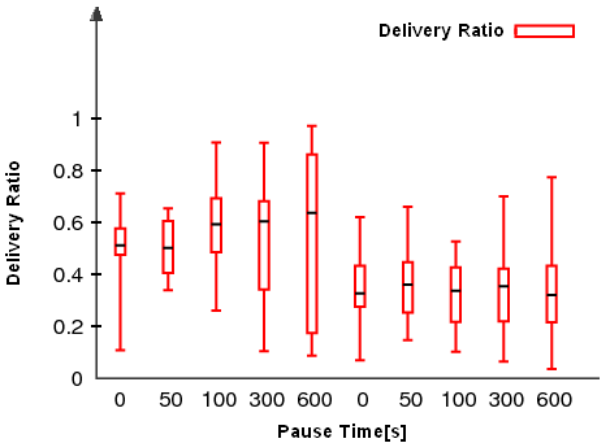


Fig. 5.1: DSR-2 Delivery Ratio (mobile app)

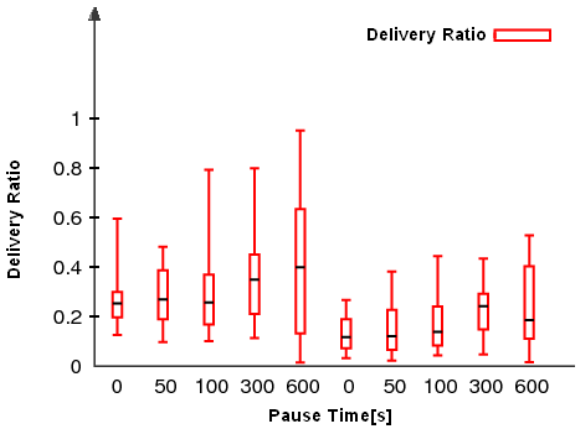


Fig. 5.2: DSR-1 Delivery Ratio (mobile app)

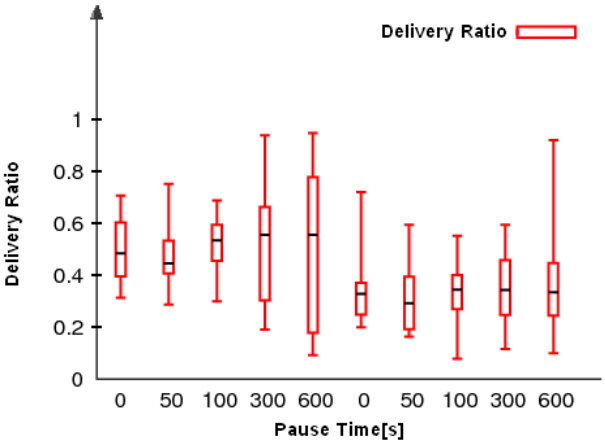


Fig. 5.3: SMR Delivery Ratio (mobile app)

X-axis represents the pause time (wait time) of all scenarios. Each column shows an individual scenario. The first range in x-aches (0s – 600s) illustrates the results by a low packet rate uniform (0.0604, 0.0608). The second range (0s – 600s) shows the results by a higher packet rate uniform (0.0202, 0.0204). The black line of each column points to the mean.

Figures 5.1, 5.2 and 5.3 show the packets delivery ratio of each protocol in mobile scenarios. We observe from the results that DSR-2 outperforms DSR-1 and SMR protocols. DSR-2 delivers up to 3% better packet delivery ratio than SMR and up to 30% better than DSR-1. One of the major problems of DSR-1 is the high traffic in the network. Because of RREPs by intermediate nodes and in processing RREQs result high network collisions. Beside, sometimes these RREPs could not be any more correct. In this case the mobile host has to wait long time to get a correct route. The disadvantage of SMR is that if all intermediate nodes forward the duplicated RREQs many collisions with other control packets or data packets may be occur as by DSR-1. Especially if there is more than one data session working at the same time, as in our case. This leads to more delay to discover new route. A lot of data packets are dropped during this time. For example: *Figures 5.4 – 5.6* shows the amount of dropped packets with 0s pause time by low packet rate and *Figures 5.7 – 5.9* shows it with 100s pause time by high packet rate.

5.2.1.2. Packet loss

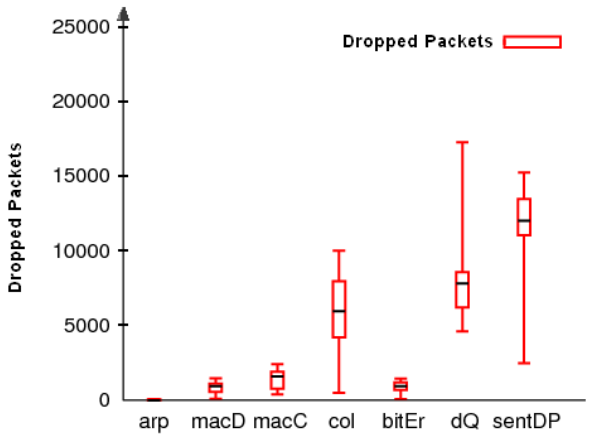


Fig. 5.4: DSR-2 Packet loss - Pause time 0 (mobile app)

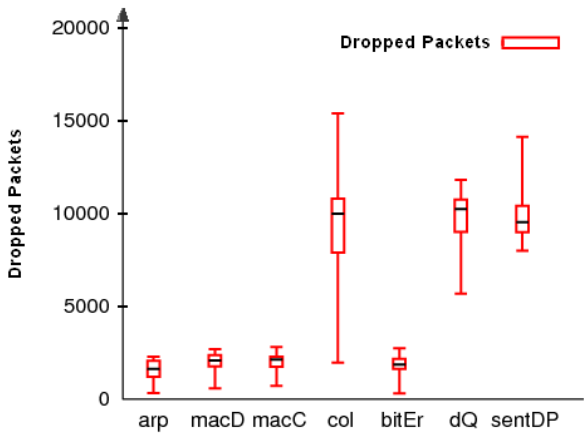


Fig. 5.5: DSR-1 Packet loss - Pause time 0 (mobile app)

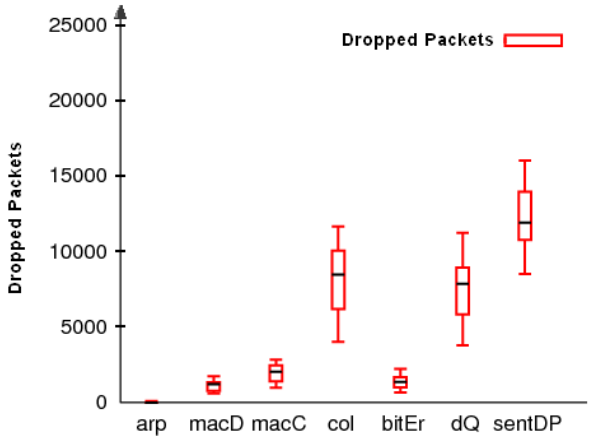


Fig. 5.6: SMR Packet loss - Pause time 0 (mobile app)

Meaning of the different column names:

arp: number of dropped data packets by ARP-Queue.
macD: number of dropped data packets by MAC-Queue
macC: number of dropped control packets by MAC-Queue
col: number of network collisions
bitEr: number of bit error packets by MAC layer
dQ: number of dropped data packets by protocol-Queue
sentDP: number of sent data packets.

The sum of dQ and sentDP packets gives the total number of sent data packets from application layer to the lower layers.

Figures 5.4, 5.5 and 5.6 show the number of dropped packets with 0s pause time by the low packet rate. DSR-2 and SMR have sent the same mount of data packets about 12400 packets of 20000 packets (*consider the main results of sentDP column*). However, SMR results in about 3000 network collisions more than DSR-2. DSR-1 has sent less data packets than DSR-2 and it results in more network collisions.

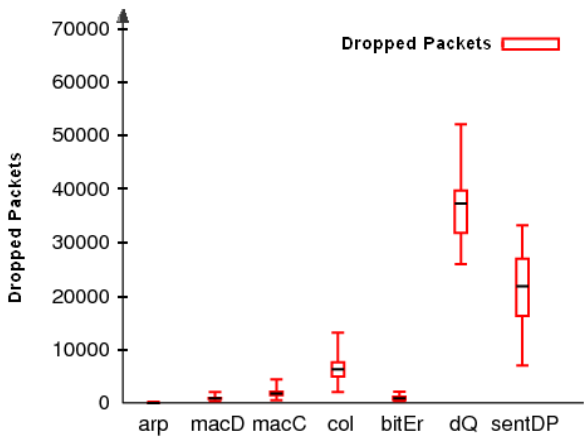


Fig. 5.7: DSR-2 Packet loss - Pause time 100 (mobile app)

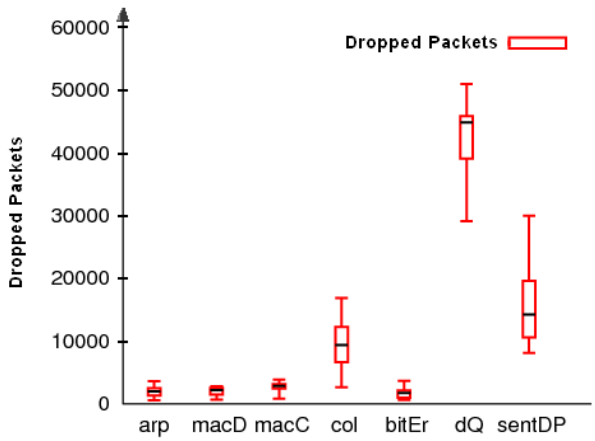


Fig. 5.8: DSR-1 Packet loss - Pause time 100 (mobile app)

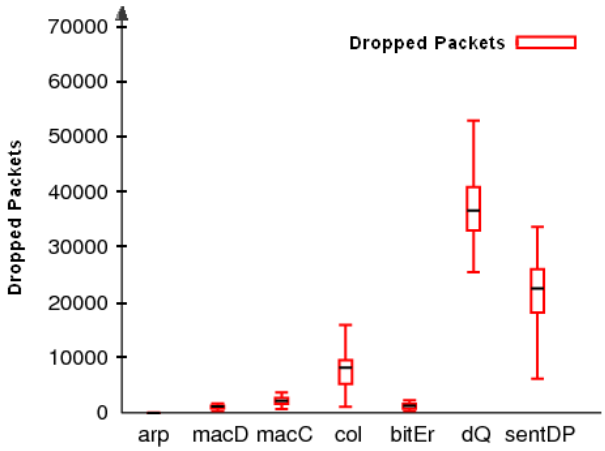


Fig. 5.9: SMR Packet loss - Pause time 100 (mobile app)

Figures 5.7, 5.8 and 5.9 show the number of dropped packets by the high packet rate and 100s pause time. DSR-2 provides better results than DSR-1 and SMR in this case. It sent more data packets with less collision in compare to DSR-1. DSR-2 and SMR have sent the same mount of data packets, but DSR-2 results less network collisions than SMR.

5.2.1.3. Data Latency, Normalized Routing Overhead and Route Live Time

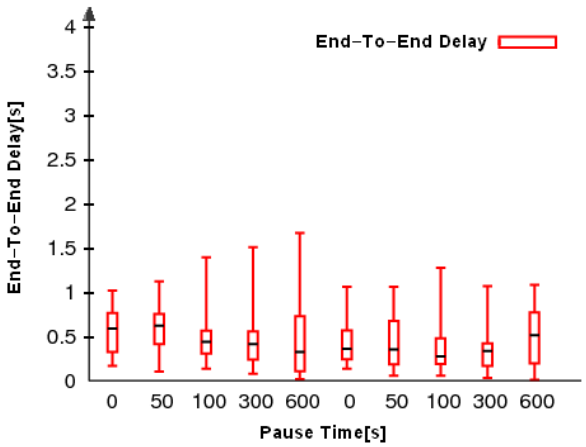


Fig. 5.10.a DSR-2 Data Latency (mobile app)

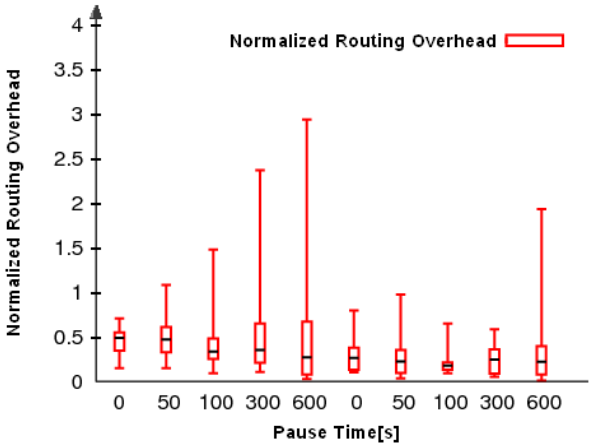


Fig. 5.10.b DSR-2 Routing Overhead (mobile app)

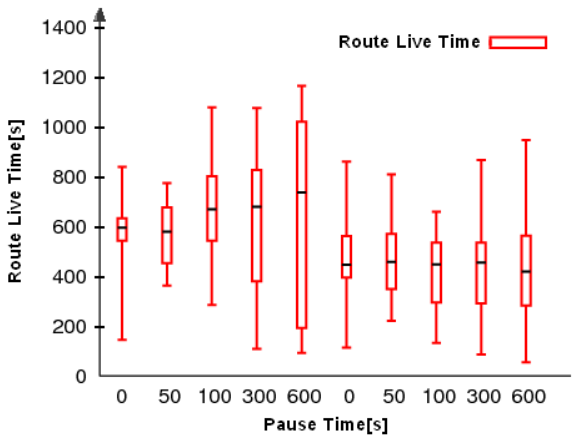


Fig. 5.10.c DSR-2 Route Live Time (mobile app)

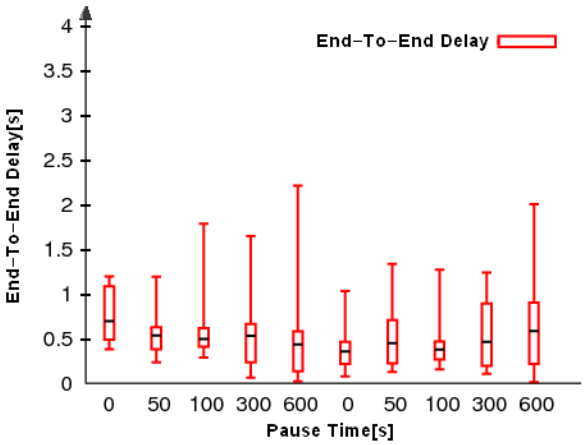


Fig. 11.a: SMR Data Latency (mobile app)

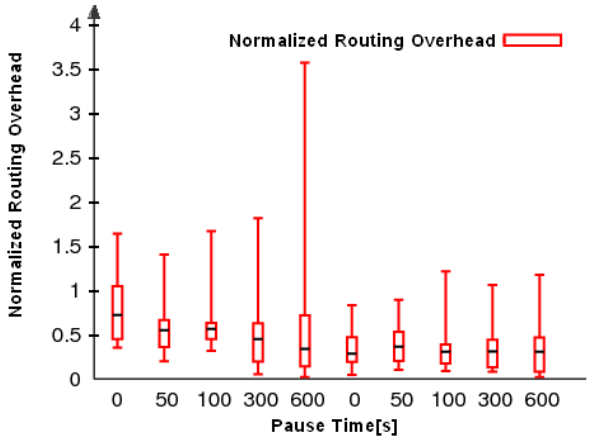


Fig. 5.11.b SMR Routing Overhead (mobile app)

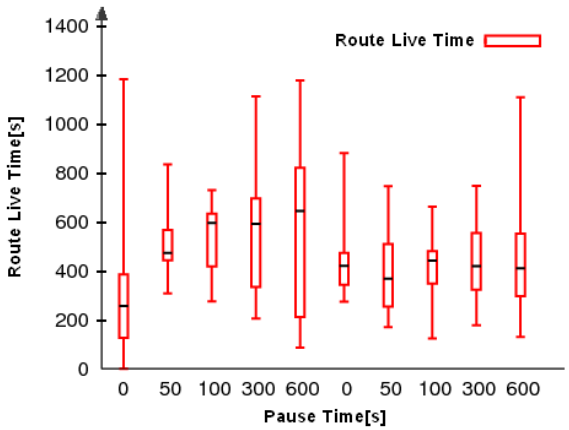


Fig. 5.11.c SMR Route Live Time (mobile app)

Figures 5.10a, 5.10b and 5.10c show better results in comparison to the *Figures 5.11a, 5.11b and 5.11c*. DSR-2 could deliver the data packets in short time and with less overhead compared to SMR for all pause times and by the both packet rate. DSR-2 showed to be more stable than SMR which was unexpected. DSR-2 outperforms the other two protocols DSR-1 and SMR.

All presented results are from mobile scenarios. In the next part, we discuss the mesh scenarios. In general mesh network showed better result for all metrics than the mobile scenarios.

5.2.2. Mesh Grid Application Scenario (mesh grid app)

The mesh grid app scenario *(Figure 5.2)* models a network of 25 mobile hosts placed randomly within an area of 1500 x 1500 meters. Each host has a transmission range of 250 meters and a channel capacity of 11MB/s. Each run executed for 600 seconds of simulation time. There are two data sessions, for which the source and destination nodes are set in the configuration file (omnetpp.ini). We use the built-in UDP application (UDPApp) of the INET framework to generate traffics with uniformly distributed packet rates in the interval [0.0604, 0.0608] for low rate scenario and in the interval [0.0202, 0.0204] for the high rate scenario. The payload size is 1450 bytes. We use use the IEEE 802.11 implementation of INET (MF80211) as the medium access control protocol. There is no mobility in this scenario (NullMobility). Several different scenarios with different host failure percents are tested. The failed hosts are set in the configuration file (omnetpp.ini).

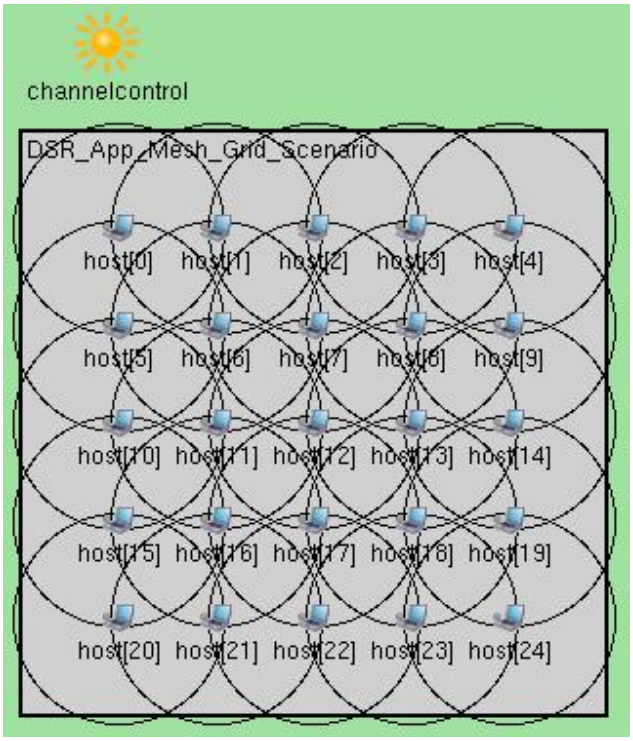


Figure 5.2 Mesh Grid Application Scenario

In the following figures, the x-axis presents the percentage of host failures by low and high packet rates.

5.2.2.1. Packet Delivery Ratio

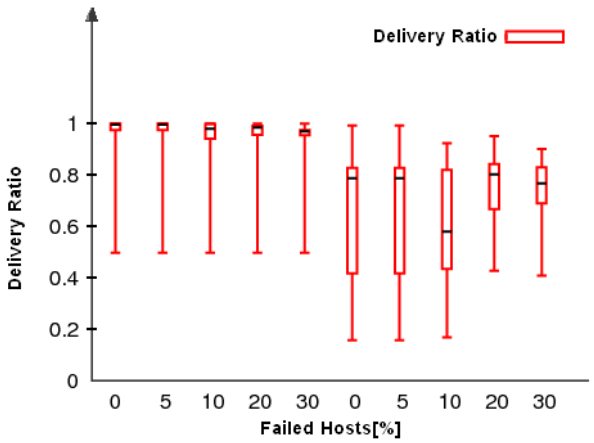


Fig. 5.12: DSR-2 Delivery Ratio (mesh grid app)

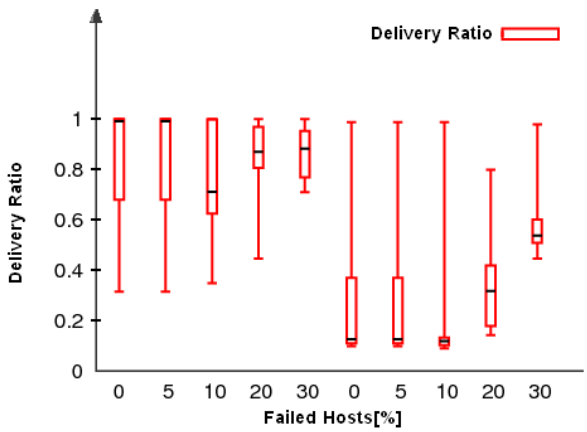


Fig. 5.13: DSR-1 Delivery Ratio (mesh grid app)

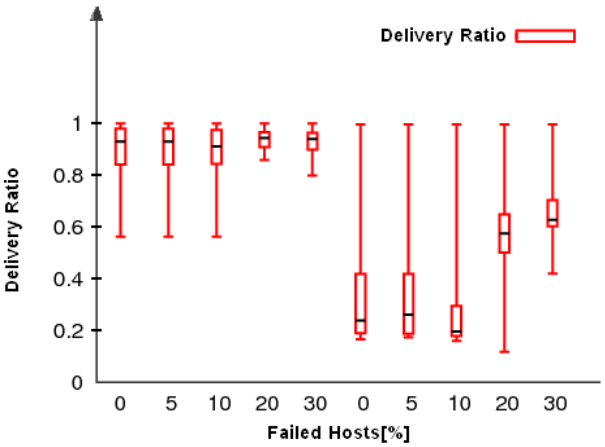


Fig. 5.14: SMR Delivery Ratio (mesh grid app)

DSR-2 shows again the best results of the both other protocols DSR-1 and SMR. DSR-1 delivers between 70% and 99% by low packet rate, but it shows to be not that stable by the high packet rate. DSR-2 shows about 99% delivery ratio by low packet rate and between 60% and 80% packet delivery ratio by high packet rate. SMR has better packet delivery ratio than DSR-1. It delivers between 94% and 97% of the sent data packets. But for high data rates, there occurs too many packet collisions. For example: *Figure 5.15* (column col) illustrates the number of packet collisions by the higher packet rate. This high number of collision leads to a very short route live time and high end-to-end delays as presented in *Figures 5.16a and 5.16c* columns 0, 5 and 10 in the second x-axis range.

5.2.2.2. Packet loss

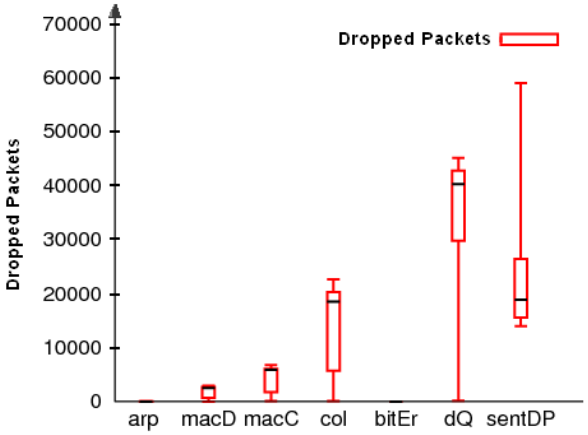


Fig. 5.15.a SMR Packet loss – Column 0 (mesh grid app)

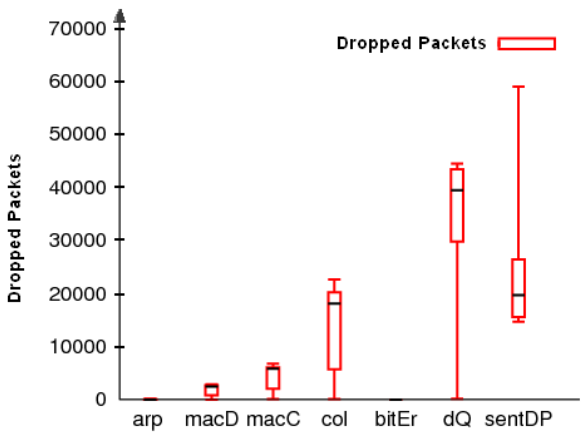


Fig. 5.15.b SMR Packet loss – Column 5 (mesh grid app)

5.2.2.3. Data Latency, Normalized Routing Overhead and Route Live Time

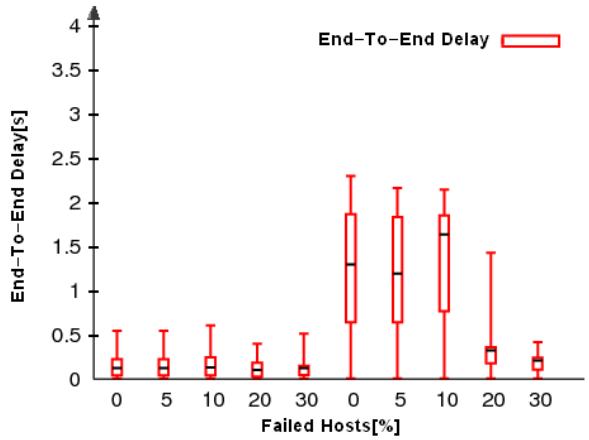


Fig. 5.16.a SMR Data Latency (mesh grid app)

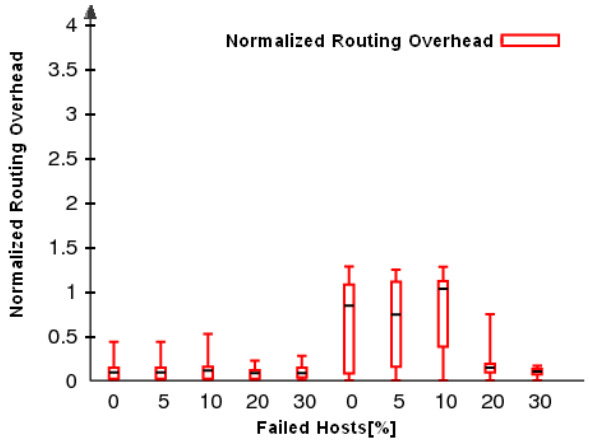


Fig. 5.16.b SMR Routing Overhead (mesh grid app)

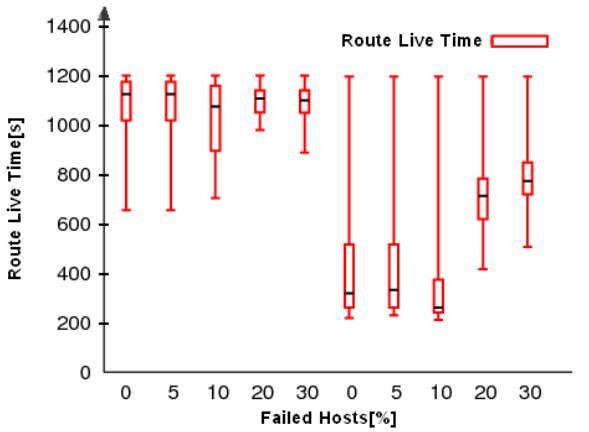


Fig. 5.16.c SMR Route Live Time (mesh grid app)

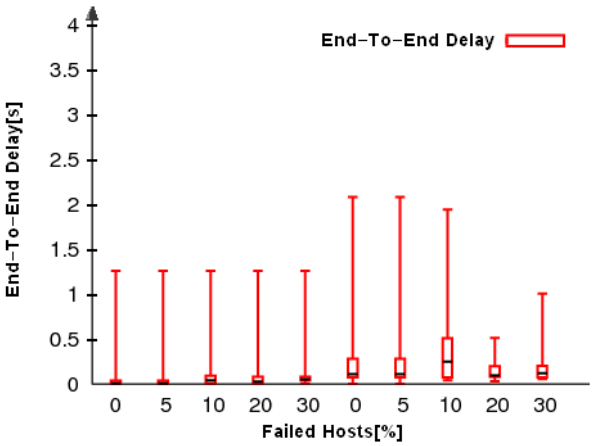


Fig. 5.17.a DSR-2 Data Latency (mesh grid app)

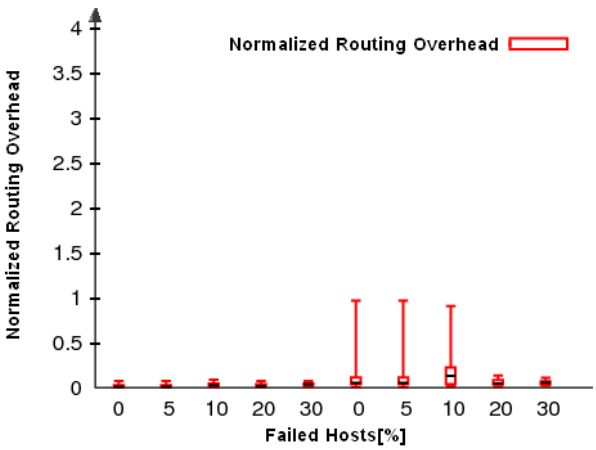


Fig. 5.17.b DSR-2 Routing Overhead (mesh grid app)

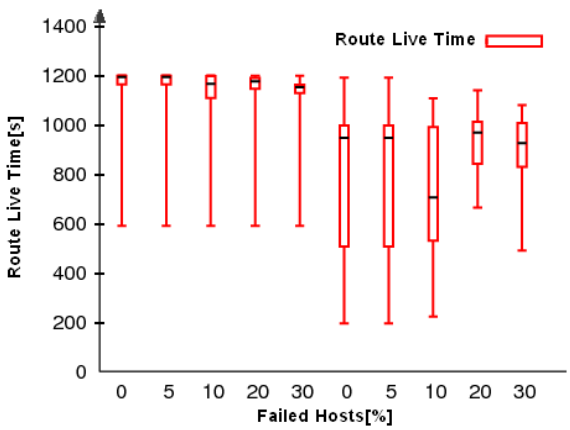


Fig. 5.17.c DSR-2 Route Live Time (mesh grid app)

Figures 5.17 show the performance of DSR-2 in mesh network scenario compared to *Figures 5.16*. DSR-2 results in very short end-to-end delays and a low routing overhead as well as a better network stability in compare to SMR and DSR-1.

To our surprise, DSR-2 outperforms the other protocols in our test scenarios for mobile and mesh networks. We thought that there should be some positive effects of the multi-path routing protocol (SMR) on the network congestion. In addition, there should be a reduced number of route dicovery processes caused by route breaks. An explanation for the results are the amount of control packets produced by the protocols and some sort of synchronization effects in the simulator. DSR-2 does not generate as many control packets as the other two protocols (DSR-1 and SMR). Thus, DSR-2 produces less congestion in the queues and the network. In general, a high number of packet collisions occur in all scenarios. This has a bad effect on our results, especially on the simulations with DSR-1 and SMR. Our investigations showed that parts of the congestion and packet collisions are due to synchronization effects in the network simulator. Therefore, the state cycle of the INET MAC layer has to be studied in more detail (see Chapter 6).

5.3. Possible Improvements

- To reduce network collision the additional feature of DSR Route Discovery "Preventing Route Reply Storms" has to be implemented [1].
- A route live timer has to be set, when a new route is saved in routing table, in order to refresh the routing table information periodically.
- The retransmission timer has to be set more randomly distributed, to avoid collisions caused by synchronization effects.

6. CONCLUSION

In this Bachelor thesis we have discussed, on the basis of several on-demand routing protocols, the two types of routing algorithm (distance vector - and source routing algorithms). And we have pointed to their advantages and disadvantages in general as well as within some example protocols.

The main routing protocol was the Dynamic Source Routing (DSR) protocol. It provides a single route per route discovery, i.e. it relies on a single route for each data session. Then we immerse oneself in study its derived multipath route protocol SMR. The Split Multipath Routing (SMR) protocol builds maximally disjoint routes, where one of them is the shortest delay route, and it uses multi routes for each data session.

The result showed that the DSR protocol, without intermediate node route reply mechanism, outperforms it's extend multipath protocol SMR. This was unexpected!

The main problem by the INET Framework is the time synchronization by resending packet from MAC layer. The packet collisions occur several times between the same packets from the same nodes. The MAC layer requires a retransmission delay mechanism (timer) using random values to avoid these systematic collisions.

List of Figures

List of Tables

REFERENCES

[1] David B. Johnson, David A. Maltz and Yih-Chun Hu, The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR), INTERNET-DRAFT, 15 April 2003, [Online]

http://www.cs.cmu.edu/~dmaltz/internet-drafts/draft-ietf-manet-dsr-09.txt

[2] D. Johnson, Y. Hu, D. Maltz, The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4, February 2007, [Online] http://tools.ietf.org/html/rfc4728

[3a] David B. Johnson, David A. Maltz, and Josh Broch, The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, a part of monarch project, [Online]

http://monarch.cs.cmu.edu/monarch-papers/dsr-chapter00.pdf

[3b] David B. Johnson, David A. Maltz, Dynamic Source Routing in Ad Hoc Wireless Networks, 1996, [Online]

http://www.cs.huji.ac.il/labs/danss/sensor/adhoc/routing/johnson_1996_dynamicsourcerouting.pdf

[4] S.-J. Lee and M. Gerla, Split multipath routing with maximally disjoint paths in ad hoc networks, in IEEE International Conference on Communications (ICC), vol. 10, (Helsinki, Finlandia), pp. 3201 3205, June 11-14 2001.

[5] R. Leung, J. Liu, E. Poon, A. L. C. Chan, and B. Li, Mp-dsr: a qos-aware multi-path dynamic source routing protocol for wireless ad-hoc networks, in 26th Annual IEEE Conference on Local Computer Networks (LCN 2001), (Tampa, Florida, USA), pp. 132 141, November 14 - 16 2001.

[6] C. Perkins and E. Belding-Royer, S. Das, Ad hoc On-Demand Distance Vector (AODV) Routing, July 2003. [Online] http://tools.ietf.org/html/rfc3561

[7] Charles E. Perkins and Elizabeth M. Belding-Royer, Ad hoc On-Demand Distance Vector (AODV) Routing, 19 January 2002, [Online] http://tools.ietf.org/html/draft-ietf-manet-aodv-10

[8] T. Clausen, C. Dearlove, J. Dean, C. Adjih, Generalized MANET Packet/Message Format, July 9 2007, [Online] http://tools.ietf.org/html/draft-ietf-manet-packetbb-08

[9] Ioannis Chatzigiannakis, Sotiris Nikoletseas, Paul Spirakis, Journal of Parallel and Distributed Computing, Volume 63, Pages: 58 – 74, 2003. [Online]

http://portal.acm.org/citation.cfm?id=782546.782553&dl=GUIDE&dl=GUIDE

[10] Omnetpp: http://www.omnetpp.org/

[11] I. Chakeres and C. Perkins. Dynamic MANET On-demand (DYMO) Routing, July 5, 2007, DYMO Draft 10, [Online] http://tools.ietf.org/html/draft-ietf-manet-dymo-10

[12] Christoph Sommer, Dynamic MANET On Demand (DYMO) routing for OMNeT++, University of Erlangen, Dept. of Computer Science 7. 1-7April 2007, [Online]

http://www7.informatik.uni-erlangen.de/~sommer/omnet/dymo/

[13] Yusuke Sakurai and Jiro Katto, AODV Multipath Extension using Source Route Lists with Optimized Route Establishment, international workshop on wireless Ad hoc network, Graduate School of Science and Engineering, Waseda University, 2004

[14] Zhenqiang Ye, Srikanth V. Krishnamurthy, Satish K. Tripathi, A Framework for Reliable Routing in Mobile Ad Hoc Networks, University of California, IEEE INFOCOM 2003

[15] Luo Liu , Laurie Cuthbert, QoS in Node-Disjoint Routing for Ad Hoc Networks, PE-WASUN'07, October 22, 2007, Chania, Crete Island, Greece.

[16] Xuefei Li and Laurie Cuthbert, On-demand Node-Disjoint Multipath Routing in Wireless Ad Hoc Networks, Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04).
[17] Xuefei Li and Laurie Cuthbert, A Reliable Node-Disjoint Multipath Routing with Low Overhead in Wireless Ad hoc Networks. Venezia, Italy MSWiM'04, October 4–6, 2004.
[18] T. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR), Project Hipercom, INRIA, October 2003, [Online] http://hipercom.inria.fr/olsr/rfc3626.txt
[19] Matthew J. Miller, Jungmin So. Improving Fault Tolerance in AODV, University of Illinois.
[20] Omnet++ version 3.2 User Manual. Kapt. 3 and 4.