Ad-hoc and Hybrid Networks

Performance Comparison of MANET Routing Protocols in Ad-hoc and Hybrid Networks

Computer Science Project

done by:
Thomas Staub
staub@iam.unibe.ch

head:
Prof. Torsten Braun

assisted by:
Marc Heissenbüttel

Computer Networks and Distributed Systems (RVS),
Institute of Computer Science and Applied Mathematics (IAM),
University of Berne, Switzerland

Februar 2004

Contents

Abstract

An ad-hoc network is an accumulation of wireless nodes forming a provisional network without any established infrastructure. Today, there exist various routing protocols for this environment. This paper compares the performance of some of them. Furthermore, the idea of extending a cellular network with ad-hoc routing facilities is haunted and the performance of some existing ad-hoc routing protocols is tested. The performance tests are done by using Network Simulator 2 (ns2). By doing the simulations the need of a routing protocol adapted to the new situation is shown.

List of Figures

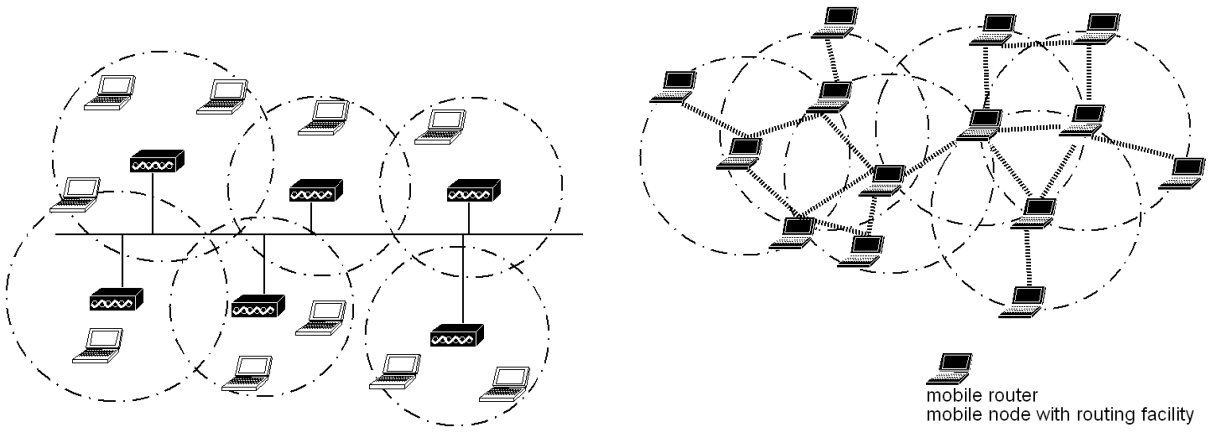
List of Tables

Listings

1 General

1.1 Mobile Adhoc Networks

A mobile ad-hoc network (MANET) is a collection of nodes, which have the possibility to connect on a wireless medium and form an arbitrary and dynamic network with wireless links. That means that links between the nodes can change during time, new nodes can join the network, and other nodes can leave it. A MANET is expected to be of larger size than the radio range of the wireless antennas, because of this fact it could be necessary to route the traffic through a multi-hop path to give two nodes the ability to communicate. There are neither fixed routers nor fixed locations for the routers as in cellular networks - also known as infrastructure networks (Fig. 1(a)). Cellular networks consist of a wired backbone which connects the base-stations. The mobile nodes can only communicate over a one-hop wireless link to the base-station, multi-hop wireless links are not possible. By contrast, a MANET has no permanent infrastructure at all. All mobile nodes act as mobile routers. A MANET is depicted in Fig. 1(b).



(a) Infrastructure Network (b) Mobile Ad-hoc Network

Figure 1: Wireless Network Structures

A MANET is highly dynamic. Links and participants are often changing and the quality of the links as well. Furthermore, asymmetric links are possible as you can see an example in Fig 2. Node A is in transmission range of node B, on the other hand node B is not in range of node A. There exist other reasons for asymmetric links such as a higher signal-to-noise ratio for node A than for node B.

1.2 Hybrid Networks

In hybrid networks the concepts of cellular networks and mobile ad-hoc networks are mixed. On one side we have a cellular network, on the other side there are mobile nodes with routing facilities. With

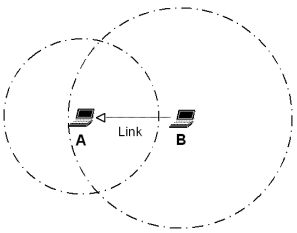


Figure 2: Asymmetric link

this approach it is possible to have multi-hop routes between mobile nodes and the base-station. The covered area of a base-station becomes larger (Fig. 3). The idea is to gain more efficiency out of the existing infrastructure, to cover wider areas with less fixed antennas and base-stations and to reduce power consumption. [1] shows the benefits of enhancing cellular network with ad-hoc technologies.

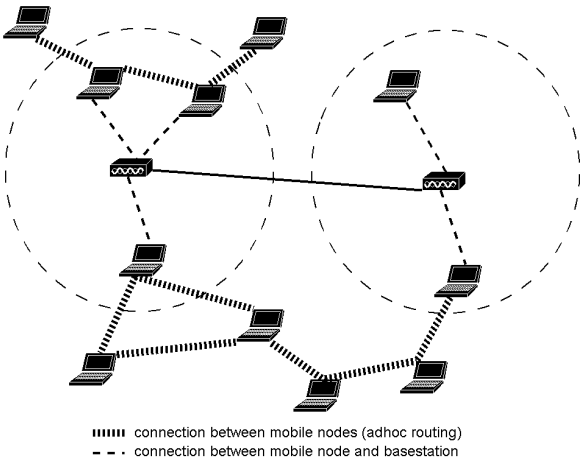


Figure 3: Hybrid network

2 MANET Protocols

New routing protocols are needed to satisfy the specific requirements of mobile ad-hoc networks. Fig. 4 gives an overview of MANET routing protocols. There exists another large family of ad-hoc routing protocols, which are location-based. Nodes must be able to determine their location, for example with GPS, to use these protocols. They are not treated in this project. Two conceptually different approaches exist for the non-location-based protocols [2]:

Table-driven (or proactive) The nodes maintain a table of routes to every destination in the network, for this reason they periodically exchange messages. At all times the routes to all destinations are ready to use and as a consequence initial delays before sending data are small. Keeping

routes to all destinations up-to-date, even if they are not used, is a disadvantage with regard to the usage of bandwidth and of network resources. It is also possible that the control traffic delays data packets, because queues are filled with control packets and there are more packet collisions due to more network traffic. Proactive protocols do not scale in the frequency of topology change. Therefore the proactive strategy is appropriate for a low mobility network.

On-demand (or reactive) These protocols were designed to overcome the wasted effort in maintaining unused routes. Routing information is acquired only when there is a need for it. The needed routes are calculated on demand. This saves the overhead of maintaining unused routes at each node, but on the other hand the latency for sending data packets will considerably increase. It is obvious that a long delay can arise before data transmission because it has to wait until a route to the destination is acquired. As reactive routing protocols flood the network to discover the route, they are not optimal in terms of bandwidth utilization, but they scale well in the frequency of topology change. Thus this strategy is suitable for high mobility networks.

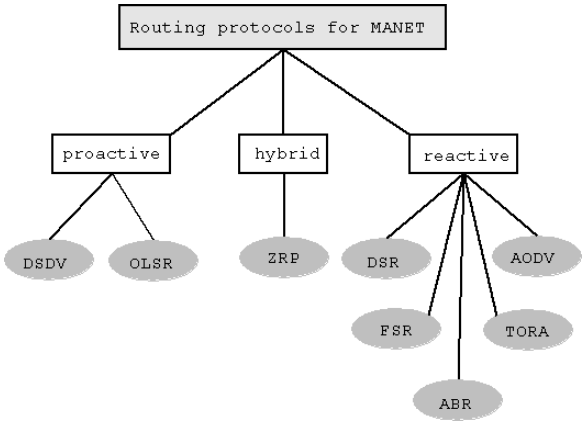


Figure 4: Classification of non-location-based ad-hoc routing protocols

2.1 Dynamic Source Routing Protocol

The Dynamic Source Routing Protocol (DSR) is a reactive routing protocol (as discussed in [3]). By the means of this protocol each node can discover dynamically a source route to any destination in the network over multiple hops. It is trivially loopfree owing to the fact that a complete, ordered list of the nodes through which the packet must pass is included in each packet header. The two main mechanisms of DSR are *Route Discovery* and *Route Maintenance*, which work together to discover and maintain source routes to arbitrary destinations in the network.

2.1.1 Route Discovery

If a node **S** wants to send to a destination node **D**, it needs a source route. **S** searches its Route Cache for a valid route to **D**. If the Route Cache contains a valid route, node **S** directly fills this route into the header of the packet and sends the data packet following this sequence of hops to the destination **D**. No Route Discovery is carried out in this case. If no route is found in the Route Cache, a Route Discovery is initiated.

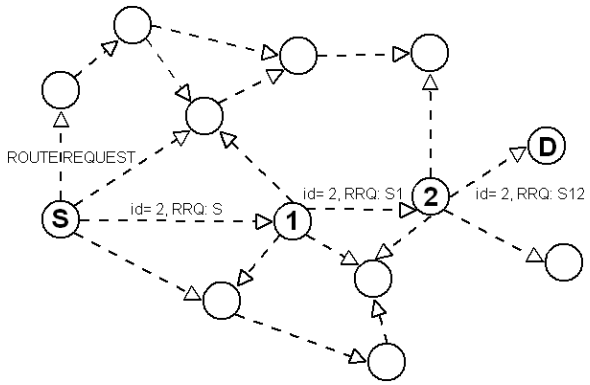


Figure 5: DSR Route Discovery example: **S** wants to discover a route to **D**

S initiates the Route Discovery by transmitting a ROUTE REQUEST message as broadcast (Fig. 5). All nodes in transmission range will receive this message. Non-target nodes will add their address to the route record in the packet and forward the packet when received the first time. They check the request id and source node id to avoid multiple retransmissions and if their address is already included in the route record to avoid loops. The target node **D** sends a ROUTE REPLY when it receives a ROUTE REQUEST. In case of bidirectional links the ROUTE REPLY uses the reversed route of the ROUTE REQUEST. If the links are unidirectional, **D** will examine its own Route Cache for a route to **S** and use this to send the ROUTE REPLY to the initiator **S**. If no route is found in Route Cache of **D**, **D** will start its own Route Discovery, but to avoid infinite numbers of Route Discoveries it will piggyback the original ROUTE REQUEST message to its own.
The initiator receiving the ROUTE REPLY adds the source route to its Route Cache. When a node starts a Route Discovery it stocks the data packet in the send buffer. The send buffer should implement strategies to avoid buffer overflows, i.e. FIFO. The node should occasionally start a new Route Discovery for the packets in the send buffer until a route is found and the packet can be sent to the destination and removed from the buffer. Due to limited transmission ranges of the nodes it is possible that some nodes are not reachable at this time. This can cause lots of useless Route Discoveries. The use of exponential back-off times between discoveries for the same target limits the wasting of resources.
In order to raise performance, a node may complement its cache with routing information it gets by overhearing or forwarding other packets. It is also possible that not only the destination node

replies to a ROUTE REQUEST, but also intermediate nodes using cached routes to the destination. Furthermore, a ROUTE REQUEST can contain a hop limit.

2.1.2 Route Maintenance

The node which originates or forwards a packet using a source route is responsible for confirming the receipt of the packet by the next node. In the situation of Fig. 5 **S** originates a packet to node **D** over the nodes **1** and **2**. In this case **S** is responsible for receipt of **1**, **1** for **2**, and finally **2** for **S**. A packet is retransmitted until a receipt is received or the number of retransmissions is exceeded. This confirmation is costless for DSR by using link-level acknowledgement frame defined by IEEE 802.11 or passive acknowledgement [4] (the forwarding of the packet to next but one from the next node is looked as a confirmation by the node).
If no confirmation is received, the node transmits a ROUTE ERROR message to the original sender indicating a broken link. The sender will remove this link from its cache and look for another source route to the destination in its cache. If the route cache contains another source route, the node sends the packet using this route. Otherwise, it will initialize a new Route Request.
In order to gain performance a node may salvage a data packet that creates the ROUTE ERROR instead of discarding it. After transmitting the ROUTE ERROR the node searches its Route Cache for a new source route. It replaces the source route in the data packet with the new source route, marks the packet to be salvaged and resends the packet. An other mechanism shortens automatically a source route when intermediate hops are no longer necessary. Additional improvements are piggybacking ROUTE ERROR information on the next ROUTE REQUEST of the node and caching negative route information (broken links are cached).

2.2 Destination Sequenced Distance Vector Protocol

The Destination Sequenced Distance Vector Protocol (DSDV) [5] is a proactive, distance vector protocol which uses the Bellmann-Ford algorithm. Compared to RIP one more attribute is added to the routing table. The sequence number as new attribute guarantees loop-freedom. It makes it possible for the mobile to distinguish stale routes from new ones and that is how it prevents loops. DSDV can only handle bidirectional links.

2.2.1 Routing Table Management

The routing table in each node consists of a list of all available nodes, their metric, the next hop to destination and a sequence number generated by the destination node. The routing table is used to transmit packets through the ad hoc network. In order to keep the routing table consistent with the dynamically changing topology of an ad hoc network the nodes have to update the routing table periodically or when there is a significant change in the network. Therefore mobile nodes advertise their routing information by broadcasting a routing table update packet. The metric of an update packet starts with metric one for one-hop neighbors and is incremented by each forwarding node and

additionally the original node tags the update packet with a sequence number. The receiving nodes update their routing tables if the sequence number of the update is greater than the current one or it is equal and the metric is smaller than the current metric. Delaying the advertisement of routes until best routes have been found may minimize fluctuations of the routing table. On the other hand the spreading of the routing information has to be frequent and quick enough to guarantee the consistency of the routing tables in a dynamic network. There exist two types of update packets. One is the full dump which contains the entire routing table and must be periodically exchanged. The other is an incremental update which only consists of the information changed since the last full dump.

2.2.2 Responding to Topology Changes

DSDV responds to broken links by invalidating all routes that contain this link. The routes are immediately assigned an infinite metric and an incremented sequence number. Broken links can be detected by link and physical layer components or if a node receives no broadcast packets from its next neighbors for a while. Then the detecting node broadcasts immediately an update packet and informs the other nodes with it. If the link to a node is up again, the routes will be re-established when the node broadcasts its routing table.

2.3 Ad-hoc On Demand Distance Vector Routing Protocol

The Ad-hoc On demand Distance Vector routing protocol (AODV) [6] joins mechanisms of DSR and DSDV. The periodic beacons, hop-by-hop routing and sequence numbers (guarantee of loop-freedom) of DSDV and the pure on-demand mechanism of Route Discovery and Route Maintenance from DSR are combined.

2.3.1 Route Discovery

Whenever there exists a valid route between two communication peers, AODV Route Discovery is not used. As soon as a route is missing between the two communication partners, e.g. when a new route to a destination is needed, a link is broken, or the route has expired, the source node **S** broadcasts a ROUTE REQUEST message in order to find a route to the destination **D**. The nodes forwarding the ROUTE REQUEST store a *reverse route* for themselves back to **S**. The ROUTE REQUEST includes the last known sequence number for that destination. It is flooded through the network until the destination **D** or an intermediate node with a valid route to **D** is reached. When the ROUTE REQUEST arrives at a node with a valid route to **D** or at **D** itself, the node sends back a ROUTE REPLY message that consists of the number of hops to **D** and the most recent sequence number. All nodes that forward this ROUTE REPLY back toward the source **S** of the ROUTE REQUEST build a *forward route* to **D**. Because of the hop-by-hop nature of AODV the nodes store only the next hop instead of the entire route. As soon as the ROUTE REPLY message arrives at the source **S**, **S** is able to send packets to the destination **D** by using the already built up *forward route*.

2.3.2 Route Maintenance

To maintain routes the nodes survey the link status of their next hop neighbors in active routes. The node detecting a link break sends a ROUTE ERROR message to each of its upstream neighbors to invalidate this route and these propagate the ROUTE ERROR to their upstream neighbors. This continues until the source node is reached. Normally the nodes in AODV sends periodic HELLO messages and the failure of reception of three consecutive HELLO messages from a neighbor is handled as link error. Another possibility of link breakage detection uses link layer notification [6]. This alternative results in a pure on-demand nature of the link breakage detection. A broken link cannot be identified until packets should be sent over the link. By contrast the HELLO messages in standard AODV allows the detection of broken links before a packet must be forwarded, but this has the disadvantage of use of bandwidth for the periodic transmission of HELLO messages. The ROUTE ERROR message contains a infinite metric for the destination and causes the receiver to invalidate the route. Now the node must start a new Route Discovery for a connection to this destination.

2.4 Optimized Linkstate Protocol

The Optimized Linkstate Protocol (OLSR) [7] is a proactive linkstate routing protocol. It uses periodic messages for updating the topology information. OLSR is based on the following mechanisms:

- neighbor sensing based on periodic exchange of HELLO messages
- efficient flooding of control traffic using the concept of multipoint relays
- computation of an optimal route using the shortest-path algorithm

2.4.1 Neighbor Sensing

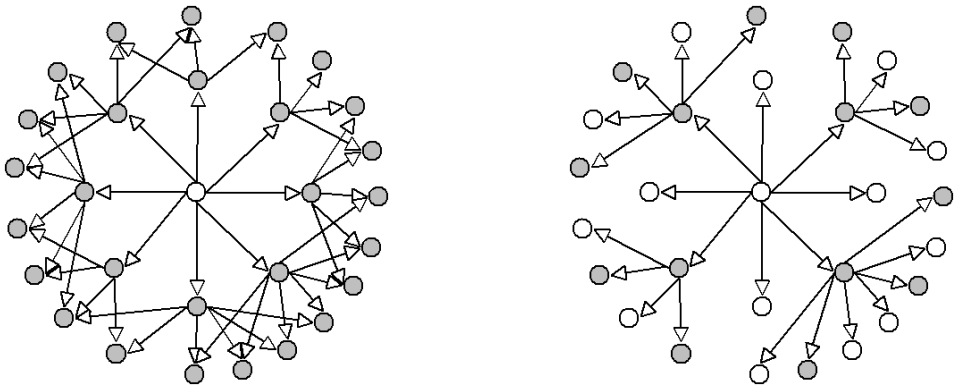
Neighbor sensing is the detection of changes in the neighborhood of the node. Node **A** is called neighbor of node **B** if the two nodes are directly linked, allowing data transmission in both directions of the link. The node **C** is called a two-hop neighbor of **A**, if node **C** is not neighbor of node and there exists a symmetric link between **A** and **B** and a symmetric link between **B** and **C**.

For neighbor sensing the node periodically emits HELLO messages. The HELLO message consists of the emitting node's address, the list of his neighbors, including the link status (e.g. asymmetric or symmetric). A node thereby informs its neighbors of which neighbors it has confirmed communication. By receiving a HELLO message, a node generates information describing its two-hop-neighborhood and the quality of the links in its neighborhood. Each node maintains this information set which is valid for a limited time only and has to be refreshed to keep it valid.

2.4.2 Message Flooding and Multipoint Relays

HELLO-messages are exchanged between neighbors only. These messages provide topology information for the nodes. Because the size of a MANET can be considerable, there is a need for efficient

distribution of topological information in a network of any size. The task is to provide a mechanism which allows spreading information to each node without unnecessary, duplicate retransmissions.
The multipoint relay (MPR) concept decreases the flooding overhead in contrast with full flooding.



(a) Pure Flooding (b) Multipoint Relays

Figure 6: Comparison of two flooding techniques

Full flooding A node retransmits broadcast packet after reception of its first copy, further duplicate receptions are dropped and not forwarded (Fig. 6(a)).

MPR flooding Each node chooses independently a set of nodes as MPRs (multipoint relays). For this purpose it utilizes the information about its two-hop neighbors to get a minimal MPR set. This set is chosen so that a node reaches all its two-hop neighbors through its MPR relays. Each nodes maintains a list of nodes which selected it as MPR (MPR selector set). A MPR node only retransmits a broadcast packet if it is received from a node for which it is located in the MPR selector set, further receptions of the same packet are dropped (Fig. 6(b)).

2.4.3 Spreading Topology Informations and Calculating Routes

Finally it is important to spread the topology information to all nodes. All nodes with a non-empty MPR selector set periodically send a topology control message (TC-message). The TC messages are spread in the network as described in 2.4.2. A TC message contains the address of its originator and the MPR set of that node. All MPRs of a node get the reachability information of that node. As a result all nodes will receive a partial topology graph by using that information and the links of their set of links to their MPR selectors. The shortest path algorithm is applied to the partial topology graph for computing the optimal path. Topology information in each node is only valid for a specific period of time and when it is expired it is removed from the graph.

3 Mobility Models

To evaluate the performance of a protocol for an adhoc network, it is necessary to test the protocol under realistic conditions, especially including the movement of the mobile nodes. A survey of different mobility models follows (cf. [8]). This includes the Random Waypoint Model that is used in this article.

3.1 Random Walk Mobility Model

This model is based on random directions and speeds. By randomly choosing a direction between *0* and 2π and a speed between 0 and V_{max}, the mobile node moves from its current position. A recalculation of speed and direction occurs after a given time or a given distance walked. The random walk mobility model is memoryless. Future directions and speeds are independent of the past speeds and directions. This can cause unrealistic movement such as sharp turns or sudden stops. If the specified time or distance is short, the nodes are only walking on a very restricted area on the simulation area.

3.2 Random Waypoint Mobility Model

A mobile node begins the simulation by waiting a specified pause-time. After this time it selects a random destination in the area and a random speed distributed uniformly between *0* m/s and V_{max}. After reaching its destination point, the mobile node waits again pause-time seconds before choosing a new way point and speed.

The mobile nodes are initially distributed over the simulation area. This distribution is not representative to the final distribution caused by node movements. To ensure a random initial configuration for each simulation, it is necessary to discard a certain simulation time and to start registering simulation results after that time.

The Random Waypoint Mobility Model is very widely used in simulation studies of MANET. As described in [9] the performance measures in mobile ad-hoc networks are affected by the mobility model used. One of the most important parameters in mobile ad-hoc simulations is the nodal speed. The users want to adjust the average speed to be stabilized around a certain value and not to change over time. They also want to be able to compare the performance of the mobile ad-hoc routing protocols under different nodal speeds. For the Random Waypoint Mobility Model a common expectation is that the average is about half of the maximum, because the speeds in a Random Waypoint Model are chosen uniformly between 0 m/s and V_{max}. But is this the average speed really reached in simulations? Not at all, the studies in [9] show that the average speed is decreasing over time and will approach 0. This could lead to wrong simulation results.

This phenomenon can be intuitively explained as follows. In the Random Waypoint Mobility Model a node selects its destination and its speed. The node keeps moving until it reaches its destination at that speed. If it selects a far destination and a low speed around 0 m/s, it travels for a long time with low speed. If it selects a speed near V_{max} the time traveling with this high speed will be short. After a certain time the node has traveled much more time at low speed than at high speed. The

average speed will approach 0 m/s. The suggestion in [9] to prevent this problem is choosing, e.g. 1 m/s instead of 0 m/s as V_{min}. With this approach the average speed stabilizes after a certain time at a value below $1/2 * V_{max}$.

3.3 Random Direction Mobility Model

To reduce *density waves* in the average number of neighbors by the Random Waypoint Model the Random Direction Mobility Model was created. *Density waves* are the clustering of nodes in one part of the simulation area. For the Random Waypoint Mobility Model the probability of choosing a location near the center or a way point which requires traveling through the center of the area is high. The Random Direction Mobility Model was invented to prevent this behavior and to promote a semi-constant number of neighbors. The mobile node selects a direction and travels to the border of the simulation area. If the boundary is reached, the node pauses for a specific time and then chooses a new direction and the process goes on. Because of pausing on the border of the area, the hop count for this mobility model is much higher than for most other mobility models.

4 Network Simulator 2 (ns2)

The Network Simulator 2 (ns2) is a discrete event driven simulator developed at UC Berkeley [10, 11]. It is part of the VINT project. The goal of ns2 is to support networking research and education. It is suitable for designing new protocols, comparing different protocols and traffic evaluations. Ns2 is developed as a collaborative environment. It is distributed freely and open source. A large amount of institutes and people in development and research use, maintain and develop ns2. This increases the confidence in it. Versions are available for FreeBSD, Linux, Solaris, Windows and Mac OS X.

4.1 Structure of ns2

ns2 is built using object oriented methods in C++ and OTcl (object oriented variant of Tcl). As you

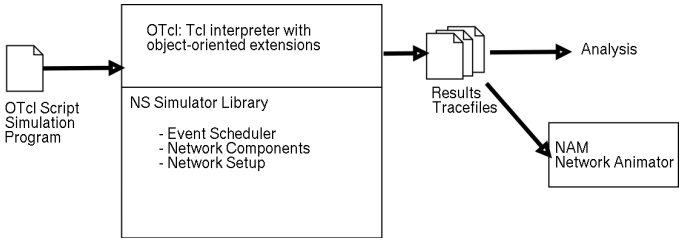


Figure 7: Simplified User's View of NS

can see in Fig. 7, ns2 interprets the simulation scripts written in OTcl. A user has to set the different components (e.g. event scheduler objects, network components libraries and setup module libraries) up in the simulation environment. The user writes his simulation as a OTcl script, plumbs the network

components together to the complete simulation. If he needs new network components, he is free to implement them and to set them up in his simulation as well. The event scheduler as the other major component besides network components triggers the events of the simulation (e.g. sends packets, starts and stops tracing). Some parts of ns2 are written in C++ for efficiency reasons. The data path (written in C++) is separated from the control path (written in OTcl). Data path object are compiled and then made available to the OTcl interpreter through an OTcl linkage (tclcl) which maps methods and member variables of the C++ object to methods and variables of the linked OTcl object. The C++ objects are controlled by OTcl objects. It is possible to add methods and member variables to a C++ linked OTcl object. A linked class hierarchy in C++ has its corresponding class hierarchy in OTcl (fig. 8). Results obtained by ns 2 (trace files, chap. 4.3) have to be processed by other tools, e.g. the Network Animator (NAM), a perl or a awk script and gnuplot [12].

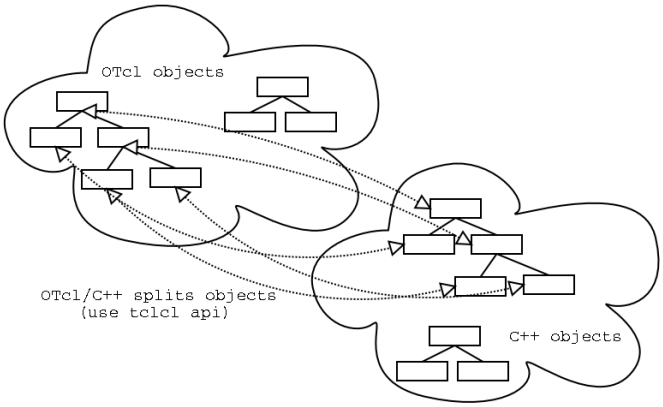


Figure 8: OTcl and C++: the duality

4.2 Functionalities of ns

Functionalities for wired, wireless networks, tracing, and visualization are available in ns2.

- Support for the wired world include
  - Routing DV, LS, PIM-SM
  - Transport protocols: TCP and UDP for unicast and SRM for multicast
  - Traffic sources: web, ftp, telnet, cbr (constant bit rate), stochastic, real audio
  - Different types of Queues: drop-tail, RED, FQ, SFQ, DRR
  - Quality of Service: Integrated Services and Differentiated Services
  - Emulation
- Support for the wireless world include

  - Ad hoc routing with different protocols, e.g. AODV, DSR, DSDV, TORA
  - Wired-cum-wireless networks
  - Mobile IP
  - Directed diffusion
  - Satellite
  - Senso-MAC
  - Multiple propagation models (Free space, two-ray ground, shadowing)
  - Energy models
- Tracing
- Visualisation
  - Network Animator (NAM)
  - TraceGraph
- Utilities
  - Mobile Movement Generator

```
setdest -n <num_of_nodes> -p pause-time -s <maxspeed>
-t <simtime> -x <maxx> -y <maxy>
```

  - Generating Traffic Patterns (CBR / TCP traffic)

```
ns cbrgen.tcl [-type cbr | tcp] [-nn nodes] [-seed seed]
[-mc connections] [-rate rate]
```

4.3 Trace Files

There exist two different trace file formats (old and new) [13, 11]. A new trace file format was introduced apart from the wireless trace format with joining the tracing in wired and wireless parts in mind.

4.3.1 Old Wireless Trace File Format

A trace in this format always begins with one of the characters in table 1. This character is succeeded by a white space separated list of values specific for the used protocol and the type of the message. For all wireless traces the values specified in the table 2 are recorded. Table 3, table 4 and table 5 contain the values for the protocols used in the simulations.

4.3.2 New Wireless Trace File Format

The structure of the new wireless trace file format is changed to be integrated in the new trace file format of the entire simulator. The lines of the trace files begin with the same action flags (table 1) as in the old format. But this is followed by flag/value pairs. The flags begin with a dash and a letter that specifies the flag type (see table 6). The trace format of a wireless event is shown in table 7. To receive a more complete list see [13].

5 Simulations

5.1 Metrics

The following metrics are often chosen to compare the different routing protocols:

Packet delivery ratio The packet delivery ratio in this simulation is defined as the ratio between the number of packets sent by constant bit rate sources (CBR, ”application layer”) and the number of received packets by the CBR sink at destination.

$$\text{packet delivery ratio} = \frac{\sum \text{CBR packets received by CBR sinks}}{\sum \text{CBR packets sent by CBR sources}} \tag{1}$$

It describes percentage of the packets which reach the destination.

Routing overhead The sum of all transmissions of routing packets sent during the simulation. For packets transmitted over multiple hops, each transmission over one hop, counts as one transmission.

$$\text{routing overhead} = \sum \text{transmissions of routing packets} \tag{2}$$

Routing overhead is important to compare the scalability of the routing protocols, the adaption to low-bandwidth environments and its efficiency in relation to node battery power (in that sending more routing packets consumes more power). Sending more routing packets also increases the probability of packet collision and can delay data packets in the queues.

s	send
r	receive
d	drop
f	forward

Table 1: These four characters specify the action that was processed to the packet.

with position	without position	**Type**	**Value**
$	$	s—r—d—f	action type
$.9d	$.9d	double	Time
$d	_$d_	int	Node ID
($6.2d		double	X Coordinate
$6.2d)		double	Y Coordinate
$3s	$3s	string	Trace Name
$4s	$4s	string	Reason
$d	$d	int	Event Identifier
$s	$s	string	Packet Type
$d	$d	int	Packet Size
[$x	[$x	hexadecimal	Time To Send Data
$x	$x	hexadecimal	Destination MAC Address
$x	$x	hexadecimal	Source MAC Address
$x]	$x]	hexadecimal	Type (ARP, IP)

Table 2: Format of a wireless event

Event	**Format**	**Type**	**Value**
ARP	- - - - - - -		
	[$s	string	REQUEST REPLY
	$i	int	Source MAC Address
	$i	int	Source Address
	$i	int	Destination MAC Address
	$i]	int	Destination Address
DSR	$i	int	Number of Nodes Traversed
	[$i	int	Routing Request Flag
	$i]	int	Route Request Sequence Number
	[$i	int	Routing Reply Flag
	$i	int	Route Request Sequence Number
	$i	int	Reply Length
	$i	int	Source of Source Routing
	->$i]	int	Destination of Source Routing
	[$i	int	Error Report Flag
	$i	int	Number of Errors
	$i	int	Report to Whom
	$i	int	Link Error From
	- >$i]	int	Link Error To

Table 3: Additional trace information according to the used protocol Part I: ARP, DSR

Path optimality The difference between the number of hops a packet took to reach the destination and the optimal number of hops to a destination (shortest path).

$$\text{path optimality} = \text{hops a packet took to reach destination} - \text{optimal number of hops} \quad (3)$$

Event	Format	Type	Value
AODV Request	[0x$x	hexadecimal	Type
	$i	int	Hop Count
	$i	int	Broadcast ID
	[$i	int	Destination
	$i]	int	Destination Sequence Number
	[$i	int	Source
	$i]	int	Source Sequence Number
	(REQUEST)		
AODV	[0x$x	hexadecimal	Type
	$i	int	Hop Count
	$i	int	Broadcast ID
	[$i	int	Destination
	$i]	int	Destination Sequence Number
	$d	double	Lifetime
	($s)	string	REPLY, ERROR, HELLO

Table 4: Additional trace information according to the used protocol Part II: AODV

Event	Format	Type	Value
IP	- - - - - - -		
	[$i:$i	int:int	Source IP Address : Port
	$i:$i	int:int	Destination IP Address : Port
	$i	int	TTL
	$i]	int	Next Hop Address
TCP	[$i	int	Sequence Number
	$i]	int	Acknowledgment Number
	$i	int	Number Of Times Packet Was Forwarded
	$i	int	Optimal Number Of Forwards
CBR	[$i]	int	Sequence Number
	$i	int	Number Of Times Packet Was Forwarded
	$i	int	Optimal Number Of Forwards
IMEP	[$c	char	Acknowledgment Flag
	$c	char	Hello Flag
	$c	char	Object Flag
	0x$04x	hexadecimal	Length

Table 5: Additional trace information according to the used protocol Part III: IP, CBR, IMEP

If there is no congestion nor other ”noise”, path optimality measures the ability to use efficiently network resources by taken the shortest path.

End to end delay The end to end delay is defined as time between the point in time the source want

N	Node Property
I	IP Level Packet Information
H	Next Hop Information
M	MAC Level Packet Information
P	Application Level Packet Information

Table 6: Flag types new wireless trace format

Flag	**Type**	**Value**
	s—r—d—f	action type
-t	double	Time
-Ni	int	Node ID
-Nx	double	X Coordinate
-Ny	double	Y Coordinate
-Ne	double	Node Energy Level
-Nl	string	Trace Name (AGT, RTR ...)
-Nw	string	Drop Reason
-Hs	int	Node ID
-Hd	int	Node ID For Next Hop
-Ma	hexadecimal	Duration
-Ms	hexadecimal	Source Ethernet Address
-Md	hexadecimal	Destination Ethernet Address
-Mt	hexadecimal	Ethernet Type
-P	string	Application Type (arp, dsr, cbr, tcp, ...)

Table 7: Wireless event using the new trace format

to send a packet and the moment the packet reaches it destination.

$$\text{end to end delay} = T_{\text{destination receives packet}} - T_{\text{source want to sent packet}} \tag{4}$$

The end to end delay is important because today many applications (e.g. ip telephony) need a small latency to deliver usable results. It shows the suitability of the protocol for these applications.

5.2 Scenario Manet

A pure Manet scenario similar to the simulations in [14] was set up in order to gain some experience and to verify the structure of the experiment. The simulation settings were as follows:

- 50 wireless nodes
- simulation area of $1500m \times 300m$. A rectangle area is chosen to have longer distances between the nodes than in a quadratic area, i.e. packets are sent over more hops.

- IEEE 802.11 MAC
- Two ray ground propagation model
- node mobility defined by random waypoint movement model.
- constant bit rate traffic
- UDP

5.2.1 Movement Model

The movement of the nodes is defined by the *random waypoint* model as described in chap. 3.2. The movement scenario files are generated by the *setdest* programm included in the ns2 distribution. The scenario files are charaterized by the *pausetime*. The simulation runs for 900 seconds of simulated time with movement patterns for seven different pause times: 0, 30, 60, 120, 300, 600, 900 seconds. A pause time of 900 seconds corresponds to a static simulation that is no motion of the nodes at all. On the contrary by selecting zero seconds as pause time there is continuous motion of the nodes. Because the performance of the routing protocols is very depended to the movements of the nodes, 10 different movement patterns for each of the seven selected pause times are generated. Each protocol is tested with the same 70 movement patterns. The *setdest* programm offers only the possibility to specify a V_{max}, V_{min} is set to zero.

5.2.2 Communication Patterns

According to the simulation in [14] the traffic sources are constant bit rate (CBR) sources. At 150s of simulated time the traffic sources start to send packets of 64 bytes. The sending rate is fixed to 4 packets per second. Three different communication patterns (one, three and ten sources) were generated. All communications are peer-to-peer in these patterns. Combining these three patterns with the 70 movement patterns (chap. 5.2.1) results in 210 different scenario files. For the pure MANET simulation all these scenario files were tested with each of the three routing protocols (AODV, DSDV, DSR).

5.3 Scenario Hybrid

The pure Manet situation is enlarged by a base station which is located at the center of the simulation area (750, 150). The simulation settings are the same as in the Manet situation (chap. 5.2). The movement model stays the same, too.

5.3.1 Communication Patterns

The communication patterns are adapted to the new situation. Three files are generated (one, three, ten connections). The base station is always one of the communication peers. Every communication is directed to the base station.

6 Conclusion

6.1 Results

There exist some stumbling blocks along the path to valid results of simulations of ns2. First of all the compilation of the simulator itself is more difficult as assumed. Some changes in the make file and in some source files were required to build ns2 on a MacOS X machine. The installation on the sun workstation at the university and on a linux computer was easier, there were only some operation system independent patches necessary. These Patches for gcc version 3.2 and for correcting DSR came from the ns2 bug fixes website and the ns2 mailing-list. The attempt to integrate the OLSR routing protocol (INRIA and NRL version) failed because of wrong ns2 versions, but also with correct versions it was not possible to get valid results. The reasons were unknown. Unfortunately the protocols DSR and DSDV did not supply any valid results in the hybrid situation. By searching the mailing-list indications were found that DSR does not work in hybrid situation. There are no reasons found for DSDV. The implementations of DSR and DSDV in ns2 need further adaption to get them work in the hybrid case but this is out of scope of this computer science project.

The results of the manet simulations are similiar to the results in [14]. Certain differences exist in the results.

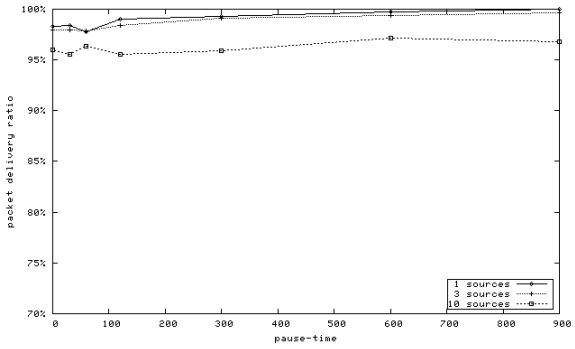
Manet AODV The curve for ten sources (Fig. 9(a)) shows a lower packet delivery ratio for pause times $>$300s than in the reference paper. Besides the routing overhead for ten sources does not converge to a small value for large pause times. The standard implementation of AODV in ns2 uses HELLO messages (as described in chapter 2.3.1) for link breakage detection instead of the pure on-demand approach (link layer) used in [14]. This causes more routing packets than using only the link layer breakage detection - even with no mobility. Further a lot of route error messages were sent (Fig. 12). No explanation for these error messages was found. The routing overhead (Fig. 11(a)) shows a similiar curve than the curve with the error messages (Fig. 12).

Manet DSDV The curve of the packet delivery ratio (Fig. 9(b)) look alike the curve in the reference paper. DSDV fails to converge below 300s pause time. By high mobility DSDV acts very badly. Because of stale routing table entries packets are sent or forwarded over broken links and packet delivery ratio breaks down for lower pause times. DSDV needs a certain time to stabilize the routes and because of this reason it is not very usable for high mobility scenarios.

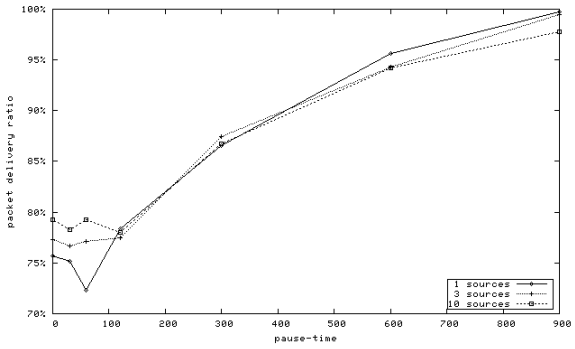
Manet DSR The packet delivery ratio for one source (Fig. 9(c)) is significantly lower than for three or ten sources. This is the effect of promiscuous listening (overhearing) of all the nodes (see

chapter 2.1.1). If there is more traffic in the network, the nodes overhear more routes and have more information about the network. On the other hand if only one source is sending packets, the knowledge of the network is limited and more packets are lost.

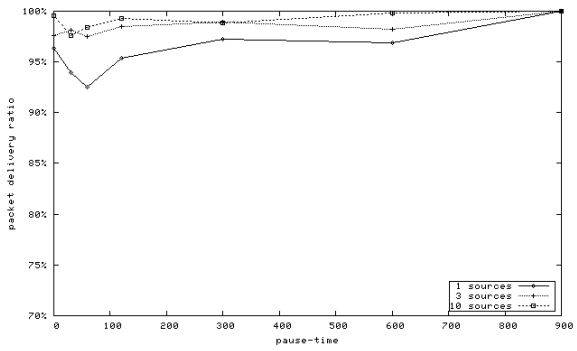
In the hybrid case no comparison of protocols was possible because of no available implementations of DSR, DSDV and OLSR which run in a hybrid scenario. AODV is performing in the same manner as in the pure manet case (Fig. 14). The same effect occurs. The packet delivery ratio breaks down for ten sources. The effect of many error messages seems to be increased by the fact of sending all messages over the base-station. Adapting the implementation of other protocols is necessary to be able to compare the performance of multiple protocols but this is out of scope of this project.



(a) AODV

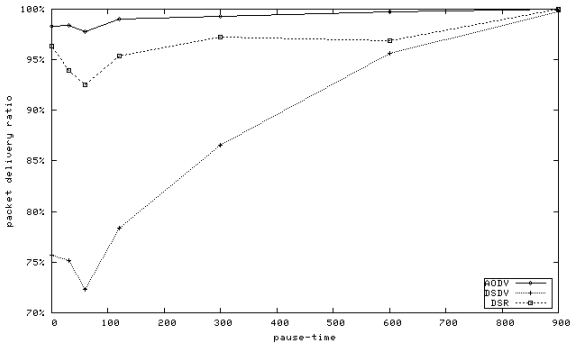


(b) DSDV

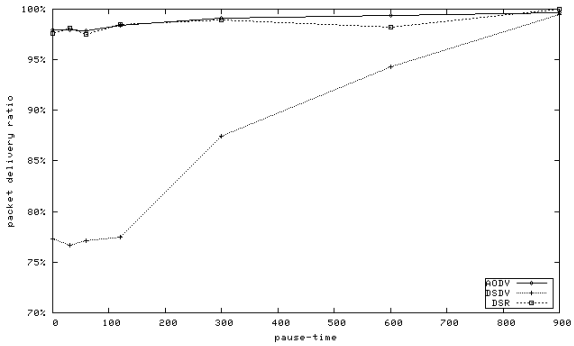


(c) DSR

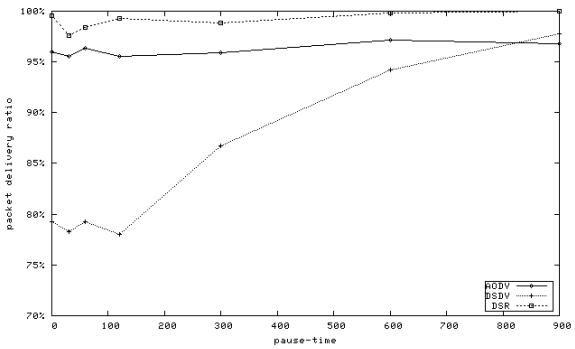
Figure 9: Packet delivery ratio with 1, 3, 5 traffic sources



(a) 1 connection

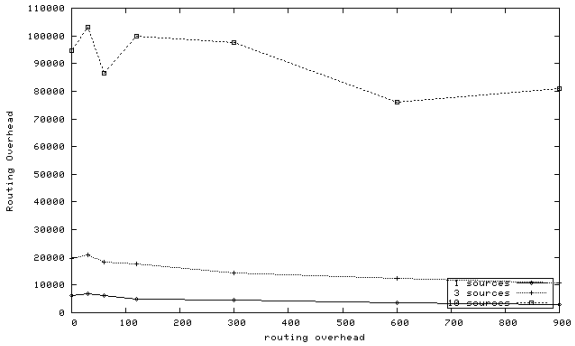


(b) 3 connections

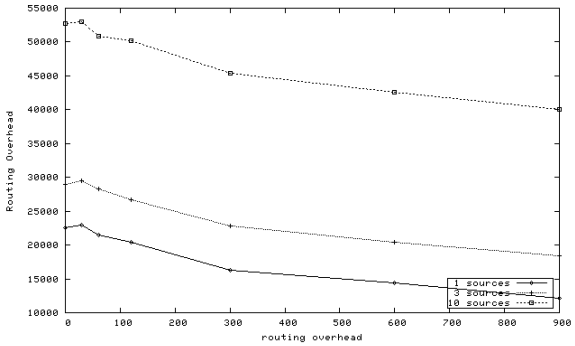


(c) 10 connections

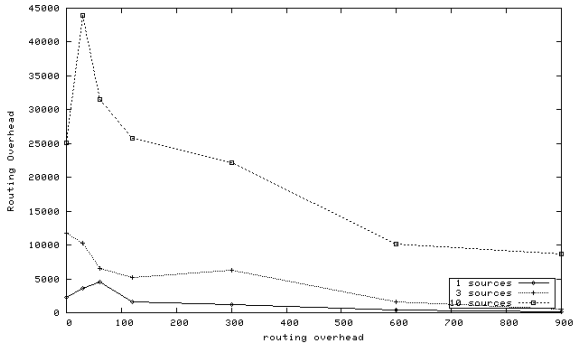
Figure 10: Packet delivery ratio with different MANET routing protocols



(a) AODV



(b) DSDV



(c) DSR

Figure 11: Routing overhead with 1, 3, 5 traffic sources

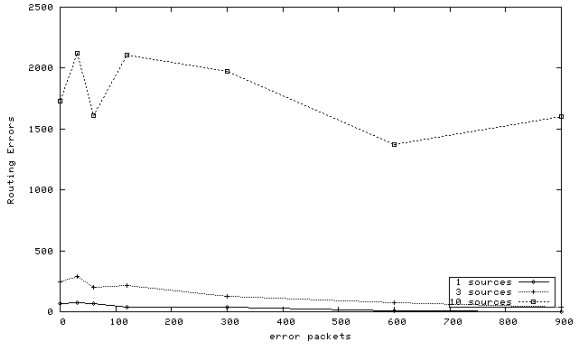
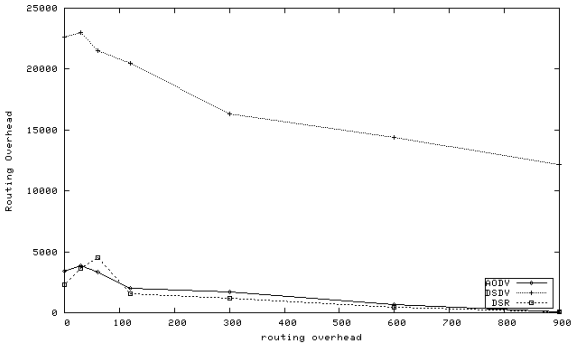
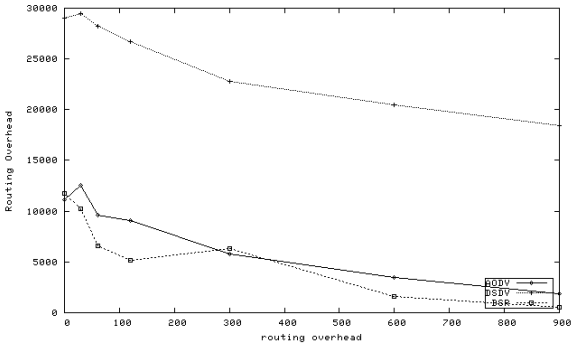


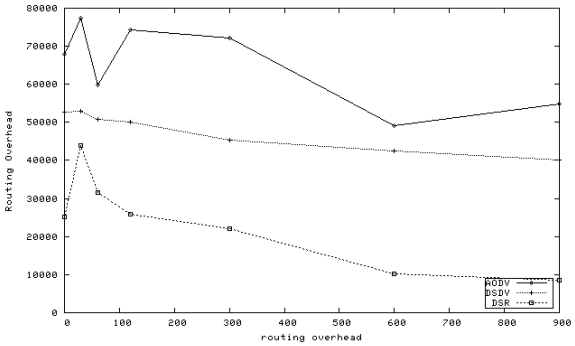
Figure 12: ERROR messages in AODV



(a) 1 connection



(b) 3 connections



(c) 10 connections

Figure 13: Routing overhead with different MANET routing protocols

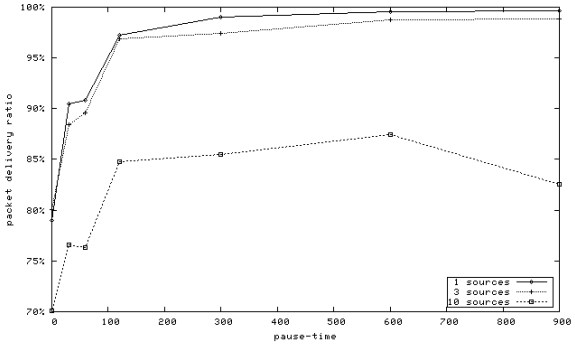


Figure 14: Packet delivery ratio with 1, 3, 5 traffic sources in a hybrid network (AODV)

6.2 Forthcoming Studies

Further investigations can be done on the following topics:

- Explanation of the sudden break-down of AODV in the manet situation. Why AODV produces a large quantity of ERROR messages?
- Simulation using AODV+, an enhanced implementation of AODV which is able to detect gateways, in the hybrid situation [15]
- Verification of the implementation of the different routing protocols
- Verification of the mobility model and the propagation model
- Verification of the 802.11 MAC layer. There exist some mails in the ns2 mailing-list that point to a possibly broken implementation of 802.11 in the ns2 version 2.26.
- Integration of power consumption. An energy model has to be defined and / or verified. The routing protocols can be tested in terms of power consumption. New routing protocols can be developed using power aware routing.
- Test other protocols in the hybrid situation. The implementations of different routing protocols in ns2 have to be adapted to use them in hybrid simulations.
- Development of a new routing protocol specialized for hybrid networks.

References

[1] H.-Y. Hsieh and R. Sivakumar, "On using the ad-hoc network model in cellular packet data networks," in *ACMMOBIHOC'02*, 2002.

[2] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad-hoc mobile wireless networks," *IEEE Personal Commun. Mag.*, vol. 6, no. 2, Apr. 1999.

[3] D. B. Johnson, D. A. Maltz, and J. Broch, "DSR: The dynamic source routing protocol for multihop wireless ad hoc networks," in *Ad Hoc Networking*. Addison-Wesley, 2001, ch. 5, pp. 139–172.

[4] J. Jubin and J. D. Tornow, "The DARPA packet radio network protocols," *Proc. IEEE*, vol. 75, no. 1, pp. 21–32, Jan. 1987.

[5] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, London, UK, Aug. 1994, pp. 234–244.

[6] C. E. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *WMCSA: 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, USA, Feb. 1999, pp. 90–100.

[7] T. Clausen, P. Jacquet, P. Muhlethaler, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol," in *IEEE INMIC'01*, Lahore, Pakistan, Dec. 2001.

[8] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002.

[9] J. Yoon, M. Liu, and B. Noble, "Random waypoint considered harmful," in *IEEE INFOCOMM 03*, San Francisco, USA, Mar. 2003.

[10] S. McCanne and S. Floyd. (2003, December) ns2 network simulator 2. [Online]. Available: http://www.isi.edu/nsnam/ns

[11] K. Fall, K. Varadhan, and the VINT project. (2003, December) The ns manual. [Online]. Available: http://www.isi.edu/nsnam/ns/ns-documentation.html

[12] (2003, December) Gnuplot 3.7. [Online]. Available: http://www.gnuplot.info

[13] R. Griswold. (2003, December) Ns-2 trace formats. [Online]. Available: http://www.k-lug.org/~griswold/NS2/ns2-trace-formats.html

[14] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, “A performance comparison of multi-hop wireless ad hoc network routing protocols,” in *Mobile Computing and Networking*, 1998, pp. 85–97.

[15] A. Hamidian, “A study of internet connectivity for mobile ad hoc networks in ns 2,” Master’s thesis, Departement of Communication Systems, Lund Institute of Technology, Lund University, Sweden, January 2003.

A Source Code

Listing 1: MANET Simulation Script

```
# shows the usage of the parameters
proc usage {} {
    global argv0

    puts "\nusage: $argv0 \[-adhocRouting AODV|DSDV|TORA|DSR|NRLOLSR\] \[-cp connectionpattern\] \[
        -sc scenario\]\[-scf folder\]"
}

proc getopt {argc argv} {
        global opt
        lappend optlist adhocRouting cp sc  scf

        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if {[string range $arg 0 0] != "-"} continue

                set name [string range $arg 1 end]
                set opt($name) [lindex $argv [expr $i+1]]
        }

}

proc handleopt {argc argv} {
global opt
# get the arguments

# read and handle parameters
getopt $argc $argv
if { $opt(adhocRouting) == "" || $opt(cp) == "" || $opt(sc) == "" || $opt(scf) == "" } {
        usage
        exit 1
}
}

Class ManetTest

ManetTest instproc init {} {
global opt val
$self instvar ns_ god_

# create simulator instance

set ns_         [new Simulator]

# setup topography object

set topo        [new Topography]

# create trace object for ns and nam

regsub -all -- {cbr_files/} $opt(cp) {} tempcp
regsub -all -- {scenario-files/} $opt(sc) {} tempsc
regsub -all -- {.cbr} $tempcp {} tempcp
regsub -all -- {.scen} $tempsc {_} tempsc
regsub -all -- {connection} $tempcp {con} tempcp

set tracefd     [open trace-files/$opt(adhocRouting)/$tempsc$tempcp-out.tr w]
```

```
$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
#
        puts $opt(adhocRouting)
        set val(ifq) Queue/DropTail/PriQueue
        if {$opt(adhocRouting) == "DSR" || $opt(adhocRouting) == "NRLOLSR"} {
                puts "DSR"
                set val(ifq)            CMUPriQueue
        }
        puts $val(ifq)
#global node setting

$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                 -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 scenario $opt(sc)"
source $opt(cp)

#
# Define traffic model
#
puts "Loading connection pattern $opt(cp)"
source $opt(sc)

#
# Tell nodes when the simulation ends
```

```
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at  $val(stop).0002 "$self finish"

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

puts "Starting Simulation..."
$ns_ run


}

ManetTest instproc finish {} {
        puts "NS EXISTING ... "
        $ns_ halt
        exit 0
}

# ======================================================================
# Define options
# ======================================================================
global opt(adhocRouting) opt(sc) opt(cp) val tempcp tempsc
set val(chan)         Channel/WirelessChannel
set val(prop)         Propagation/TwoRayGround
set val(netif)        Phy/WirelessPhy
set val(mac)          Mac/802_11
set val(ll)           LL
set val(ant)          Antenna/OmniAntenna
set val(x)              1500   ;# X dimension of the topography
set val(y)              300    ;# Y dimension of the topography
set val(ifqlen)         50            ;# max packet in ifq
set val(seed)           1.0
set opt(adhocRouting)  ""
set val(nn)             50             ;# how many nodes are simulated
set opt(sc)             ""
set opt(scf)                    ""
set opt(cp)             ""
set val(stop)           900.0           ;# simulation time
set tempcp              ""
set tempsc              ""
        handleopt $argc $argv
        set test [new ManetTest]
```

B Source Code Hybrid

Listing 2: Hybrid Simulation Script

```
# shows the usage of the parameters
proc usage {} {
    global argv0

    puts "\nusage: $argv0 \[-adhocRouting AODV|DSR|OLSR\] \[-cp connectionpattern\] \[-sc scenario
        \]\[-scf folder\]"
```

```
}

# read the parameters
proc getopt {argc argv} {
        global opt
        lappend optlist adhocRouting cp sc scf

        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if {[string range $arg 0 0] != "-"} continue

                set name [string range $arg 1 end]
                set opt($name) [lindex $argv [expr $i+1]]
        }

}

# ======================================================================
# Define options
# ======================================================================

set val(chan)         Channel/WirelessChannel
set val(prop)         Propagation/TwoRayGround
set val(netif)        Phy/WirelessPhy
set val(mac)          Mac/802_11
#set val(ifq)          ""
set val(ll)           LL
set val(ant)          Antenna/OmniAntenna
set val(x)              1500   ;# X dimension of the topography
set val(y)              300    ;# Y dimension of the topography
set val(ifqlen)         50             ;# max packet in ifq
set val(seed)           1.0
set opt(adhocRouting)  ""
set val(nn)             50              ;# how many nodes are simulated
set opt(sc)             ""
set opt(scf)                    ""
set opt(cp)             ""
set val(stop)           900.0            ;# simulation time
set tempcp              ""
set tempsc              ""

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

#
# Initialize Global Variables
#
# get the arguments

# read and handle parameters
getopt $argc $argv
if { $opt(adhocRouting) == "" || $opt(cp) == "" || $opt(sc) == "" || $opt(scf) == ""} {
        usage
        exit
}


# create simulator instance

set ns_         [new Simulator]
# set up for hierarchical routing
```

```
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2          ;# number of domains (wired and wireless
lappend cluster_num 1 3               ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 11 20 20             ;# number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain

# setup topography object

set topo       [new Topography]

# create trace object for ns and nam

regsub -all -- {cbr_files/} $opt(cp) {} tempcp
regsub -all -- {scenario-files/} $opt(sc) {} tempsc
regsub -all -- {.cbr-bs} $tempcp {} tempcp
regsub -all -- {.scen} $tempsc {_} tempsc
regsub -all -- {connection} $tempcp {con} tempcp

set tracefd     [open "trace-files/hybrid/$opt(adhocRouting)/$tempsc$tempcp-out.tr" w]


$ns_ trace-all $tracefd

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

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



puts "$opt(adhocRouting)"

set ifqueue Queue/DropTail/PriQueue
if {$opt(adhocRouting) == "DSR" || $opt(adhocRouting) == "NRLOLSR"} {
               set ifqueue  CMUPriQueue
}
puts "Used queue: $ifqueue"

set W(0) [$ns_ node 0.0.0]

#
# define how node should be created
#


# sets node configuration for basestation node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $ifqueue \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                                 -topoInstance $topo \
                 -wiredRouting ON \
                                  -agentTrace ON \
```

```
                -routerTrace ON\
                -macTrace OFF

#create the base-station node
set temp {1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8 1.0.9 1.0.10 1.1.11 1.1.12 1.1.13 1.1.14 1
    .1.15 1.1.16\
                            1.1.17 1.1.18 1.1.19 1.1.20 1.1.21 1.1.22 1.1.23 1.1.24 1.1.25 1
                                .1.26 1.1.27 1.1.28 1.1.29 1.1.30 1.2.31 1.2.32\
                            1.2.33 1.2.34 1.2.35 1.2.36 1.2.37 1.2.38 1.2.39 1.2.40 1.2.41 1
                                .2.42 1.2.43 1.2.44 1.2.45 1.2.46 1.2.47\
                            1.2.48 1.2.49 1.2.50}   ;# hier address to be used for wireless
                                domain
set BS(0) [$ns_ node 1.0.0]
$BS(0) random-motion 0               ;# disable random motion

#sets the basestation to the middle of the grid
$BS(0) set X_ $val(x)/2.0
$BS(0) set Y_ $val(y)/2.0
$BS(0) set Z_ 0.0


#configure for mobilenodes
$ns_ node-config -wiredRouting 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 [lindex $temp [expr $i]]]
        $node_($i) random-motion 0              ;# disable random motion
    $node_($i) base-station [AddrParams addr2id [$BS(0) node-addr]]
}

$ns_ duplex-link $W(0) $BS(0) 5Mb 2ms DropTail
$ns_ duplex-link-op $W(0) $BS(0) orient left-down

#
# Define connections
#
puts "Loading connection pattern..."
source $opt(cp)

#
# Loads movement model
#
puts "Loading scenario file..."
source $opt(sc)

#
# Propagate the stop of Simulation to the nodes
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}

$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 $opt(adhocRouting)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"
```

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

Listing 3: Running simulation for all scenario files

```
#!/usr/bin/perl -w
my @cpfiles=<cbr_files/*.cbr>; #We now have those files in the array @cp-files
my @folders=("4","5","6","7","8","9");
my @protocols=("DSDV");
foreach $proto (@protocols) {
        foreach $folder (@folders){
                my @scfiles=<scenario-files/$folder/*s20.scen>;
            foreach $cpfile (@cpfiles) {
                foreach $scfile (@scfiles){
                system("ns manet_test.tcl -adhocRouting $proto -sc $scfile -scf $folder -cp
                    $cpfile");
                print "\nns manet_test.tcl -adhocRouting $proto -sc $scfile -scf $folder -cp
                     $cpfile durchgefuehrt";
                }
            }
    }
}
```

Listing 4: Generating Routing Overhead Plots

```
#!/usr/bin/perl -w

# Thomas Staub
# project student
#
# University of Berne, Switzerland

my $numberOfRoutingPackets = 0;
my $file = "testfile";
my @pausetimes = (0,30,60,120,300,600,900);
my $speed = 20;
my @sources = (1,3,10);
my @protocols = ("AODV", "DSDV","DSR");
my @folders = ("0","1","2","3","4","5","6","7","8","9");

foreach $proto (@protocols) {


    foreach $source (@sources) {
            $output = "gnuplot/roh_" . $proto . "_s" . $speed . "_con" . $source;
            open(OUTPUT, "> $output");
            print OUTPUT "# routing overhead\n";
            print OUTPUT "# $proto speed $speed sources $source \n";
            print OUTPUT "# pause-time | routing overhead\n";

            foreach $pausetime (@pausetimes) {

                $numberOfRoutingPackets = 0;

                foreach $folder (@folders) {
                    $file = "trace-files/".$proto . "/".$folder."/node50_p" . $pausetime
                        . "_s" . $speed . "_" . $source .  "con-out.tr";
                    open(FILE,"< $file") or die "Could not open file $file : $!";
                    while (defined ($line =<FILE>)) {
                        chomp $line;
```

```
                                        # count the routing packets
                                        if($line =~ /s.*RTR.*/){
                                                $numberOfRoutingPackets++;
                                        }
                                }
                        }
                        $numberOfRoutingPackets = $numberOfRoutingPackets/@folders;
                        close(FILE) or die "Could not close file $file : $!";
                        print OUTPUT "$pausetime  \t " . $numberOfRoutingPackets . " \n";

                }
                close(OUTPUT);
        }
        #generates a gnuplot for one protocol and all numbers of sources
        $gnuplot = "gnuplot << !\nset terminal png monochrome\nset ylabel \"Routing Overhead\"\nset
            xlabel \"routing overhead\"\nset output \"images/manet/". $proto . "_roh_sources.png\"\
            nset key right bottom box\nset size 1.0,0.8\nplot ";
        for ($i=0; $i <= $#sources;$i++){
                unless($i== $#sources){
                        $gnuplot .= "\"gnuplot/roh_" . $proto ."_s" . $speed . "_con" . $sources[$i
                            ] . "\" title '" .$sources[$i] ." sources' with linespoints,  ";
                }
                else {
                        $gnuplot .= "\"gnuplot/roh_" . $proto ."_s" . $speed . "_con" . $sources[$i
                            ] . "\" title '" .$sources[$i] ." sources' with linespoints\n!";
                }
        }
        system($gnuplot);

}

#generates a gnuplot, that compares all protocols with 1,3,10 connections
foreach $source (@sources) {
        $gnuplot = "gnuplot << !\nset terminal png monochrome\nset ylabel \"Routing Overhead\"\nset
            xlabel \"routing overhead\"\nset output \"images/manet/comparison_con".$source."_roh.png
            \"\nset key right bottom box\nset size 1.0,0.8\nplot ";
        for ($i=0; $i <= $#protocols;$i++){
                unless($i== $#protocols){
                        $gnuplot .= "\"gnuplot/roh_" . $protocols[$i] . "_s" . $speed . "_con"
                            .$source."\" title '" . $protocols[$i] . "' with linespoints, ";
                }
                else{
                        $gnuplot .= "\"gnuplot/roh_" . $protocols[$i] . "_s" . $speed . "_con"
                            .$source."\" title '" . $protocols[$i] ."' with linespoints\n!";
                }
        }
        system($gnuplot);
}
```

Listing 5: Generating Packet Delivery Plots

```
#!/usr/bin/perl -w

# Thomas Staub
# project student
#
# University of Berne, Switzerland

my $sent = 0;
my $received = 0;
```

```
my $file = "testfile";
my @pausetimes = (0,30,60,120,300,600,900);
my $speed = 20;
my @sources = (1,3,10);
my @protocols = ("AODV","DSDV","DSR");
my @folders = ("0","1","2","3","4","5","6","7","8","9");

foreach $proto (@protocols) {


        foreach $source (@sources) {
                $output = "gnuplot/pdr_" . $proto . "_s" . $speed . "_con" . $source;
                open(OUTPUT, "> $output");
                print OUTPUT "# packet delivery ratio\n";
                print OUTPUT "# $proto speed $speed sources $source \n";
                print OUTPUT "# pause-time | packet delivery ratio\n";

                foreach $pausetime (@pausetimes) {

                        $sent = 0;
                        $received = 0;

                        foreach $folder (@folders) {
                                $file = "trace-files/".$proto . "/".$folder."/node50_p" . $pausetime
                                     . "_s" . $speed . "_" . $source .  "con-out.tr";
                                open(FILE,"< $file") or die "Could not open file $file : $!";
                                while (defined ($line =<FILE>)) {
                                        chomp $line;

                                        #count the sent packets
                                        if($line =~ /s.*AGT.*cbr.*/){
                                                $sent++;
                                        }

                                        # count the received packets
                                        if($line =~ /r.*AGT.*cbr.*/){
                                                $received++;
                                        }
                                }
                        }
                        $received = $received/@folders;
                        $sent = $sent/@folders;
                        close(FILE) or die "Could not close file $file : $!";
                        print OUTPUT "$pausetime  \t " . (($sent != 0)? $received / $sent: 0 )*100 .
                             " \n";

                }
                close(OUTPUT);
        }
        #generates a gnuplot for one protocol and all numbers of sources
        $gnuplot = "gnuplot << !\nset terminal png monochrome\nset ylabel \"packet delivery ratio\"\
            nset format y \"%.0f%%\"\nset yrange [70:100]\nset xlabel \"pause-time\"\nset output \"
            images/manet/". $proto . "_pdr_sources.png\"\nset key right bottom box\nset size 1.0,0.8
            \nplot ";
        for ($i=0; $i <= $#sources;$i++){
                unless($i== $#sources){
                        $gnuplot .= "\"gnuplot/pdr_" . $proto ."_s" . $speed . "_con" . $sources[$i
                            ] . "\" title '" .$sources[$i] ." sources' with linespoints, ";
                }
                else {
                        $gnuplot .= "\"gnuplot/pdr_" . $proto ."_s" . $speed . "_con" . $sources[$i
                            ] . "\" title '" .$sources[$i] ." sources' with linespoints\n!";
```

```
                }
        }
        system($gnuplot);

}

#generates a gnuplot, that compares all protocols with 1,3,10 connections
foreach $source (@sources) {
        $gnuplot = "gnuplot << !\nset terminal png monochrome\nset ylabel \"packet delivery ratio\"\
            nset format y \"%.0f%%\"\nset yrange [70:100]\nset xlabel \"pause-time\"\nset output \"
            images/manet/comparison_con".$source."_pdr.png\"\nset key right bottom box\nset size 1
            .0,0.8\nplot ";

                for ($i=0; $i <= $#protocols;$i++){
                unless($i== $#protocols){
                        $gnuplot .= "\"gnuplot/pdr_" . $protocols[$i] . "_s" . $speed . "_con"
                            .$source."\" title '" . $protocols[$i] . "' with linespoints, ";
                }
                else{
                        $gnuplot .= "\"gnuplot/pdr_" . $protocols[$i] . "_s" . $speed . "_con"
                            .$source."\" title '" . $protocols[$i] ."' with linespoints\n!";
                }
        }
        system($gnuplot);
}
```