Universität Karlsruhe (TH)

Performance Comparison of flat and self-organized routing protocols for MANETs

Studienarbeit

Institut für Telematik
Prof. Dr. M. Zitterbart
Fakultät für Informatik
Universität Karlsruhe (TH)

und

Laboratoire CITI, INRIA ARES Project
Département Télécommunications, Services & Usages
Institut National des Sciences Appliquées de Lyon

von

cand. inform.
Denis Martin

Betreuer:
Prof. Dr. M. Zitterbart
Fabrice Valois, Maître de Conférences PhD.
Fabrice Theoleyre, PhD. Student - Ing.
Dipl.-Inform. Peter Baumung

Tag der Anmeldung:	14. März 2005
Tag der Abgabe:	13. September 2005

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 13. September 2005

Contents

1. Introduction

Nowadays, wireless communication is already widespread, even in the private sector: Mobile phone cellular networks nearly have the same importance as the 'old' wired telephone networks. Computers also started to communicate over a wire. But becoming smaller and smaller, they also became portable and at the latest, when they reached the size of handhelds, they were truly mobile. But differently to cellular phones, there was not immediately the need to connect such mobile devices to a national or international network, but primarily, one wanted to connect to an office or home LAN independently of available wires and independently of the location within the office.

Thus, wireless LANs came up: Mobile devices were equipped with radio transceivers and an access point was installed within the office that acted as a gateway towards the wired office LAN. But this mode of wireless IP communication supported by an infrastructure, although already widespread, is not the only one. Wireless LANs may also be established spontaneously between two or more mobile devices. Such mobile ad-hoc networks are self-organized and decentral without any central instance managing the communication. Since no infrastructure has to be installed, such networks are surely interesting for military purposes, but also for civil deployment like in catastrophe management to support coordination of emergency teams or just for private use like Instant Messaging, Voice over IP or even multiplayer games – all without the need of any interconnection equipment that has to be installed.

But there is still recent research going on, because there is one crucial point: routing. While multihop routing already exists when it comes to the interconnection of LANs to WANs and to the Internet, routing over multiple radio hops is different. The medium is much less reliable than in wired networks and the participating devices are mobile and have different capacities. The topology underlies high dynamics, thus, routes between nodes may change very often.

1.1 Objectives of this document

The objective of this work is the comparison of two routing protocols for mobile ad-hoc networks that are using different approaches. One of these protocols is the

Ad-hoc On-demand Distance Vector (AODV) routing protocol. AODV is already accepted by the Internet Engineering Task Force (IETF) and provides a reactive on-demand route construction, that is, the route between two nodes is build up only when there is data to be sent by the user. The other protocol is the Virtual Structure Routing (VSR) protocol. Its approach is based on a self-organization of the network. Rather to consider a flat network like AODV, VSR deals with a structured network. Furthermore, VSR combines both, proactive (i.e. routes are constructed and maintained continuously) and reactive routing.

VSR was already compared with another protocol, that also uses a hybrid approach like VSR: the Cluster Based Routing Protocol (CBRP, [ThVa05b]). As a current work, a comparison between VSR and a purely proactive protocol, the Optimized Link State Routing (OLSR) protocol, is made. Finally, this work compares VSR with a purely reactive protocol: AODV.

The testing environment for comparing these representatives of the different genre of routing protocols has to be the same. Thus, they were all implemented for the same simulation environment starting from their original documentation (RFC, if possible). Furthermore, the implementations of mobility and traffic generation models were reused for every protocol to achieve exactly the same surrounding conditions.

1.2 Structuring

At the beginning, a brief overview of wireless networks is given in chapter 2. It explains some terms used within this document as well as some introductory words concerning routing in mobile ad-hoc networks.

Chapter 3 gives an overview of the AODV protocol and then describes it more in detail. Control packet formats and control structures used by AODV are given. Furthermore, it describes how AODV constructs routes and how it keeps track of existing routes that are used.

VSR is explained in chapter 4. It starts with the basic ideas upon which VSR's virtual structure is based. After this, the topology construction and maintenance procedures are described. Finally, the routing upon this structure is explained.

The implementation of AODV in OPNET Modeler Version 8.1.A is described in chapter 5. It also explains the implementation of the traffic and mobility models used and documents the possible simulation parameters.

Chapter 6 shows the configuration parameters that were used for the simulation series. After that, the results are discussed and explained.

Finally, this work concludes in chapter 7. Here, a short review of this work is given and some possible future workings, that are based on this one, are pointed out.

2. Basic principals

This chapter describes some basic knowledge needed for the understanding of upcoming chapters. In section 2.1 a general introduction to wireless LANs is given, including medium access. After that, in section 2.2, some mechanisms to find routes between nodes in mobile ad-hoc networks are explained.

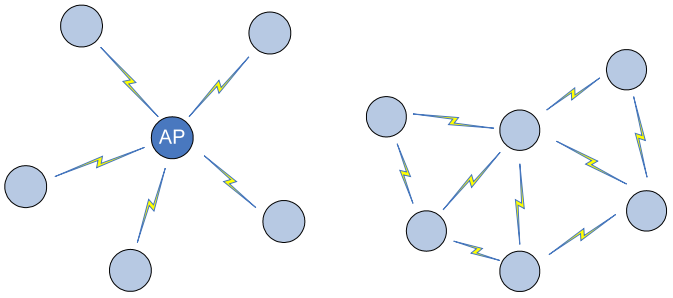
2.1 Mobile ad-hoc networks

Basically, wireless networks can be organized in two different ways: with an *infrastructure* and without an infrastructure. In the first case, there is a base station present, the *access point* (AP), and all communication goes through it (figure 2.1(a)). The access point often is connected to a wired LAN and may provide access to services like printing and Internet in that way. Thus, this *managed* mode of operation is essentially an extension of a wired network to easily provide flexible access to an existing infrastructure via radio.

In operation without such an infrastructure, the nodes are communicating directly with each other, in *ad-hoc* mode (figure 2.1(b)). When adding the ability to communicate via multiple intermediate nodes through multihop routing protocols (see section 2.2), entire networks emerge. Within such *mobile ad-hoc networks* (MANETs), communication is not limited to nodes that are within radio range of each other. This allows nodes willing (or needing) to communicate with each other to establish a network anywhere without the need to install any interconnection equipment. While this form of communication may be interesting for military units to exchange strategy information during a maneuver, it also may have a wide civil deployment, for example in civil protection on catastrophes, allowing emergency units to coordinate, or in traffic management systems allowing a fast propagation of traffic jam information from car to car.

2.1.1 Hybrid networks

A limitation of a managed wireless LAN is, that a node has to be within radio range of the access point. However, when communicating in ad-hoc mode, a node may



(a) Managed network (b) Ad-hoc network

Figure 2.1: Types of a wireless LANs

still reach its desired destination node even if it is far beyond radio range, as long as there are enough intermediate nodes that forward the data.

Hybrid networks are actually a combination of mobile managed and ad-hoc networks (figure 2.2). Mobile nodes are communicating like in ad-hoc mode. The access point also behaves like such a node, except that it doesn't move and that it has an additional network interface. This interface is a standard Ethernet interface that is connected to a wired network. Thus, the access point has a gateway functionality between two IP networks.

For now, there is no standardization for hybrid networks yet. But they are becoming more and more interesting also regarding terms like Mobile IP and Ubiquitous Internet, since they allow access to a wired network for nodes that are not within radio range of an access point.

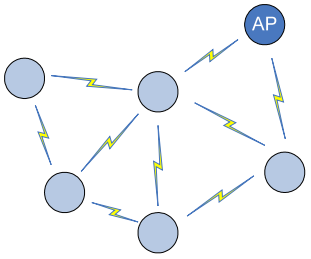


Figure 2.2: Hybrid network

2.1.2 Medium access: IEEE 802.11

In radio communication, medium access techniques that are used for wired LANs like CSMA/CD (carrier sense multiple access with collision detection, 'listen before and while talk') are not suitable ([Tane03]): While all nodes on an Ethernet bus are aware of ongoing communications, this is not the case for radio communication. Consider figure 2.3: When A is transmitting to B and C senses the medium, it will not hear A because it is out of radio range. Consequently, if C wants to transmit to B, it starts transmitting, interferes at B and garbles the transmission of A. The medium was free at the sender, but not at the receiver. A was too far away from C to be detected by C, thus hidden. This problem is called the *hidden station problem.*

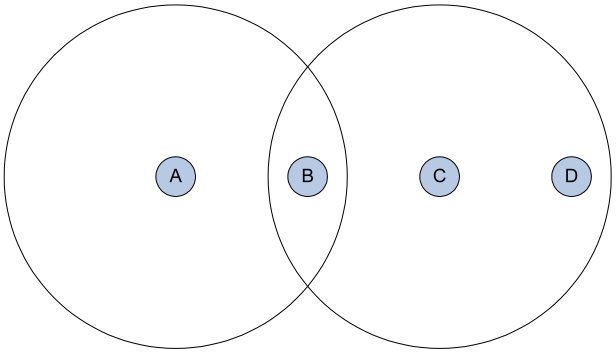


Figure 2.3: Hidden station problem: A is hidden to C and vice versa

Another problem occurs if B transmits to A and C wants to transmit to D (figure 2.4). If C senses the medium, it will hear the ongoing communication of B and therefore defer its transmission to D, although there wouldn't be any interference at the receiver. So C is exposed to B but B is far away enough to D. This problem is called the *exposed station problem*.

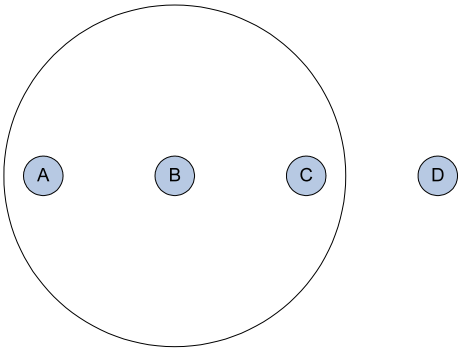


Figure 2.4: Exposed station problem: C is exposed to B

The *Institute of Electrical and Electronic Engineers* (IEEE) defined a standard for wireless LANs that comprises managed and ad-hoc modes and that is compatible with existing networking protocols above the data link layer: the IEEE 802.11 standard ([Schi03]). This standard defines the physical layer (2.4 GHz radio band) and the data link layer (direct communication between mobile stations or with a base station). Thus, in terms of 802.11, 'ad-hoc' means a direct communication between stations that are within radio range. This is contrary to mobile ad-hoc networks that also imply multihop communication (see introduction of section 2.1).

In ad-hoc mode, the CSMA/CA (CSMA with collision avoidance) protocol is used for medium access control. In CSMA/CA, a node A that wants to transmit data to node B first senses the medium in listening a certain time to it (*DCF InterFrame Spacing*, DIFS). If it is occupied by another transmission, A waits a certain backoff time and retries again. If it was not occupied during the time DIFS, A starts its data transmission. B is supposed to send an *acknowledgment* upon successful reception of the data frame. If A does not receive any acknowledgment within a certain timeout, it assumes that its transmission collided with a transmission of

another node. Colliding nodes wait a random time that is calculated using a binary exponential backoff algorithm and then try again later.

This basic access to the medium is still vulnerable to the hidden station problem. The RTS/CTS extension of CSMA/CA attempts to avoid collisions due to hidden stations: When the medium finally becomes free, A sends a *RTS* (request to send) frame to B to ask for permission to send it a data frame. If B receives such a RTS and if it is free for data reception, it replies with a *CTS* (clear to send) frame. Both frames are very short and contain the size of the data frame that is to be sent. If A does not receive the expected CTS, it waits a random backoff time and then starts a retry. If it receives the CTS successfully, it sends the data frame.

Hidden stations learn about attempted transmissions even if only one concerned node, the sender or the receiver, is within radio range. If a third party node receives a RTS, it estimates the duration of the upcoming transfer and marks the medium busy for that time. Either does a node that receives a CTS. The waiting time can be estimated using the size of the upcoming data frame that is transmitted within the RTS and CTS frames. Thus, all nodes that are within the radio ranges of either A or B are informed about the data transmission and will keep quiet during the transfer.

2.2 Routing in MANETs

While routing protocols in wired networks are designed for fixed topologies with fixed neighborhoods and only few changes, routing in MANETs is very different: Nodes may disappear suddenly, reappear somewhere else, have unidirectional links towards other nodes and move away from one neighbor towards a new neighbor while keeping any third neighbors. Thus, routes in wired networks may be valid for a long time while routes in MANETs may become invalid instantly after construction.

In MANETs there are two basic types of routing protocols that try to cope with the high dynamics in two different ways:

- Reactive routing protocols

  Reactive protocols do not maintain any persistent information about the network in which they are participating. If there is no communication at all, the route table of a node normally remains empty and a node does not know any of its neighbors. In fact, only if a higher protocol layer wishes to send data to a particular node, the routing protocol starts its work and enters a *route discovery* process, thus it starts route construction *on-demand*. Usually, route requests are broadcasted and flooded on the network until there is some route reply. Unused routes are normally discarded from the route table after a short period of time because they are not maintained actively and, thus, must not be considered valid after a certain time due to the high dynamics of MANETs. Examples for reactive routing protocols for MANETs that are currently being standardized by the IETF are the Dynamic Source Routing (DSR) protocol ([JoMH04]) and the Ad-hoc On-Demand Distance Vector (AODV) protocol (RFC 3561, see chapter 3).

- Proactive routing protocols

  With proactive protocols, nodes continuously exchange routing information and a node usually has a valid route to each other node within the network. The advantage is, that there is no latency due to route construction like with reactive protocols. The tradeoff, however, is usually a constant base network load due to route information exchange even if there is no user data to be sent. This also results in a higher energy consumption. Examples for proactive routing protocols for MANETs that are currently being standardized by the IETF is the Optimized Link State Routing (OLSR) protocol (RFC 3626, [CJLM+01]) and the Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) protocol (RFC 3684, [OgTL04]).

Reactive protocols usually have the advantage that the control flow related to routing is reduced. However, they have to pay with a relatively high latency for data transmission when constructing a route on-demand. Proactive protocols allow data to be sent almost always immediately, but they usually produce more control flow and consume more energy and memory which can be critical to small low power devices like PDAs. Some protocols combine reactive and proactive routing in order to find a good compromise between control flow and route construction latency. They are often hierarchical routing protocols. Examples are the Cluster Based Routing Protocol (CBRP, [JiLT99]) and Virtual Structure Routing (VSR, see chapter 4).

3. The AODV protocol

The Ad hoc On-Demand Distance Vector (AODV) Routing Protocol [PeBRD03] is intended for the Internet Protocol (IP) to provide a routing protocol adapted to mobile nodes in an ad hoc network. It is a reactive protocol that builds routes between two nodes only if a communication between these two nodes is desired.

This chapter describes the functionality of AODV, starting from a description of the basic principle and going more into detail when describing special treatment in certain cases that should improve its performance.

3.1 Overview

For route discovery and management, AODV uses four basic control packets. The *RouteRequest*-packet is broadcasted by an origin node if it doesn't have a route to a destination node to which it desires to send a data packet. This packet is relayed in broadcast by all neighboring nodes receiving it and forwarded further through the network until it reaches a node that knows the requested destination, possibly the destination itself. This node then generates a *RouteReply*-packet as a positive reply which is forwarded in unicast back on the reverse path that was created while forwarding the *RouteRequest*. These two packets are used by the source, the destination and all intermediate nodes on the route to create appropriate route table entries. The *RouteError*-packet is sent if an intermediate node on the route cannot relay a data packet for a specific node (see figure 3.1). Finally, the route reply acknowledgment (*RouteReplyAck*) packet has to be sent by the originating node when it receives a *RouteReply* with an appropriate flag set. This option is intended to assure the creation of bidirectional links when there is a danger of unidirectional links due to different radio ranges among the intermediate nodes.

The AODV protocol also supports multicast routing and group management ([RoPe00]). Using the *RouteRequest*/*RouteReply* mechanism, a node detects multicast tree members. With a special *MulticastActivation* packet unicasted to the nearest group member, it finally joins the group.

Control packets of AODV are sent to port 654 using UDP. So an AODV control packet is encapsulated by an UDP header and an IP header. Some information from

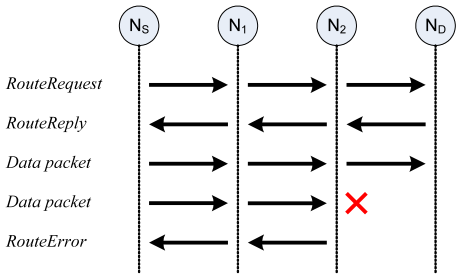


Figure 3.1: This shows an example of a route discovery that is answered by the requested destination itself and a link breakage detected by node N_2

these headers, like the address of the previous hop or the time to live of a packet (`TTL`), are also used by the AODV protocol.

3.2 Formats and structures

This section describes the packet formats of the main control packets. Furthermore, the necessary information of AODV's route table and some brief notes about its maintenance are given. A detailed description of the use of each field will be given later in sections 3.3 and 3.4.

3.2.1 Packet formats

Figure 3.2 shows the packet formats of the four main control packets. The `Type` field identifies the packet type: *RouteRequest*, *RouteReply*, *RouteError*, *RouteReplyAck* or *Hello*[1]. The `Hop Count` field represents the number of hops taken between the originating and the destination node. The `Destination IP Address` always refers to the IP address of the node for which a route is requested, whilst the `Originator IP Address` stands for the emitter of a *RouteRequest*. The sequence numbers are the associated control packet sequence numbers of the respective node.

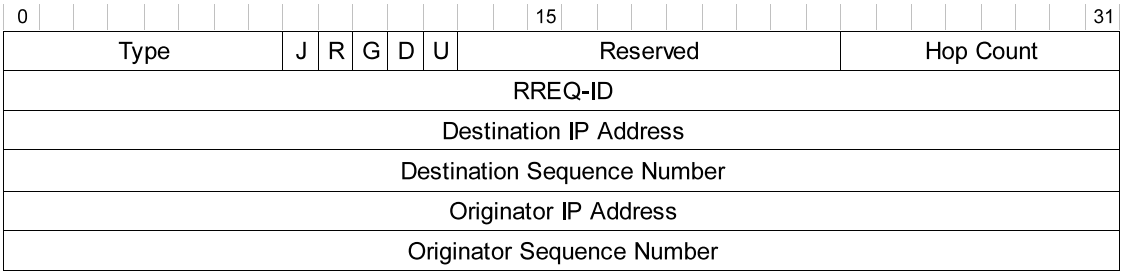
The `RREQ-ID` is a sequence number for *RouteRequests*. It is incremented for each *RouteRequest* emitted by the node, also for route request retries if a previous *RouteRequest* timed out. In contrast to that, the `Originator Sequence Number` is only incremented once for each route discovery cycle.

Within the *RouteRequest* packet, the following flags may be set:

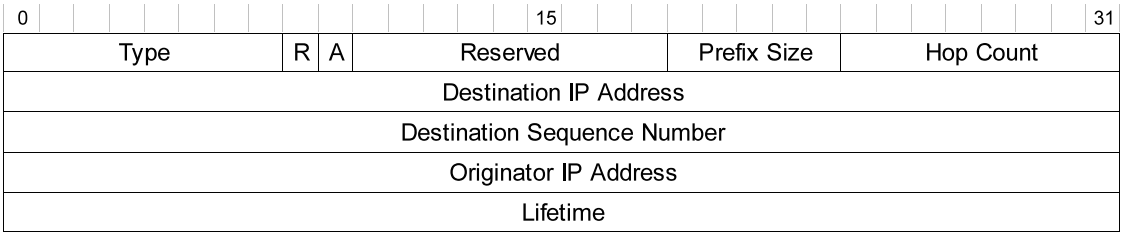
- `J` - group join flag (for multicast purposes)
- `R` - repair flag (for multicast purposes)
- `G` - demand gratuitous route reply (see section 3.3.2)
- `D` - destination only flag: only the destination may reply to this request, no intermediate replies are wanted
- `U` - destination sequence number unknown: indicates that the originator doesn't know the destination's sequence number and that the respective field in the *RouteRequest* packet should be ignored

[1] see section 3.4.2

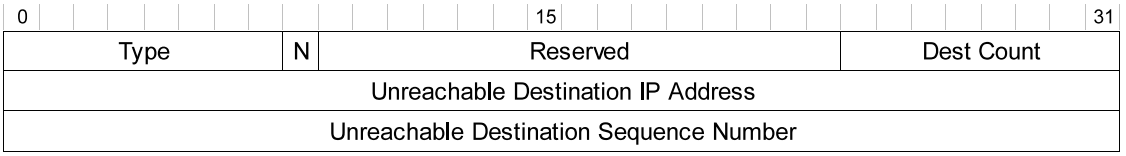
RouteRequest



RouteReply



RouteError



RouteReplyAck

Figure 3.2: Format of AODV's control packets

A *RouteReply* packet may have the following flags:

- `R` - repair flag (for multicast purposes)
- `A` - acknowledgment required: the recipient (the next hop) has to send a *RouteReplyAck* to the previous hop

Finally, the `N` flag (no delete flag) of a *RouteError* packet indicates that a unreachable route should not be deleted because a local repair of a link has been performed.

The *RouteReplyAck* contains no further information, so it is impossible to determine which *RouteReply* is acknowledged by that packet. However, there is no real need to identify a particular packet since, with this mechanism, the node only verifies if a link to a neighbor is bidirectional or not.

3.2.2 Route table format

An AODV route table entry contains at least the following information:

`destAddr`	the destination node's IP address for which the entry is for
`destSeqNo`	its last known sequence number
`destSeqNoValid`	boolean, indicates if the `destSeqNo` is known and therefore valid
`routeValid`	boolean, indicates if the `nextHopAddr` for that entry is valid
`lifetime`	time when this entry expires; see note below
`nextHopAddr`	if `routeValid`, this indicates the next hop to reach that node
`hopCount`	number of hops to the destination node
`precursors`	list of precursors that use us as an intermediate hop to the destination node

The `lifetime` field has two purposes: If the route table entry is valid, it indicates the time when the route to that node expires. When it expired, the `routeValid` flag is set to false and the lifetime field is set to `DELETE_PERIOD`; so from this point, it indicates the time when the entry has to be deleted from the route table. Thus, after a route is invalid, it still remains for `DELETE_PERIOD` in the route table. This is necessary since some information may be reused when re-requesting a route to the same destination node: the last known `hopCount` is used as a starting radius for the expanding ring search (see section 3.3.1) and the `destSeqNo` may be used in the *RouteRequest* packet as additional information.

The list of precursors contains the addresses of nodes that have a route to `destAddr` on which we are as an intermediate, forwarding node. This information is used in case we have to generate or forward a *RouteError* packet (see section 3.3.3).

3.3 Control Flow

In this section, the whole process of AODV's route construction is explained. The last subsection shows, how AODV deals with broken routes.

3.3.1 Route discovery

If a source node wishes to send a data packet to a specific destination, it first looks in its route table if it has a valid route to the destination IP address. If it finds such an entry, it simply unicasts the data packet to the next hop indicated in the route table entry (`RTE`). Additionally, it sets the life time of the `RTE` to current time + `ACTIVE_ROUTE_TIMEOUT` to prolong the life time of the entry because it has been used again.

If such an entry does not exist, a *RouteRequest*-packet will be broadcasted. To prevent flooding the whole network unnecessarily, an expanding ring search technique is used. The first dissemination of a *RouteRequest* will have a `TTL` of `TTL_START`, so that only the next `TTL_START` hops are affected by the broadcasted packet. If the origin node does not receive a *RouteReply* within a certain timeout (`RING_TRAVERSAL_TIME`), it increases the `TTL` of the *RouteRequest*-packet by `TTL_INCREMENT`, increments its `RREQ-ID` and resends the *RouteRequest*. This is repeated until a *RouteReply* is received or until the `TTL` reaches `TTL_THRESHOLD`. Note that the value of `RING_TRAVERSAL_TIME` takes the current `TTL` as a parameter. Thus, with an increasing ring, the waiting time also increases.

After this and if the origin node has not yet received a *RouteReply*, it sends a *RouteRequest* with `TTL` set to `NET_DIAMETER` hops. This may be repeated up to `RREQ_RETRIES` times, but each time the waiting time is increased using a binary exponential backoff. Hence, for every new *RouteRequest*, the waiting time is multiplied by 2.

The value `RREQ-ID` within a *RouteRequest*-packet represents together with the emitter's node address a unique identifier of a particular *RouteRequest*. It is incremented by the emitter for each new *RouteRequest* sent. This is necessary to identify duplicates of a request while flooding the respective packet through the network.

3.3.2 Processing route requests

When a node receives a *RouteRequest*, it creates an active route table entry (`RTE`) for the previous hop if necessary and updates the life time field of that `RTE` to current time + `ACTIVE_ROUTE_TIMEOUT`. Then, the node verifies if it has already received that particular *RouteRequest* (identified by its origin address and its `RREQ-ID`). In this case, it just discards the *RouteRequest*.

If it is the first time, that the node receives the request, the hop count of the *RouteRequest* is incremented by one. The node also updates its `RTE` for the origin node (creating it if necessary) for the reverse route:

- The originator's sequence number is copied from the *RouteRequest* packet to the appropriate `RTE` if the new one is greater or if it was previously unknown.
- As next hop to the originator, the IP address of the node is set, from which we received the *RouteRequest*. The address is obtained from the IP header of the packet.
- The hop count is copied from the *RouteRequest* packet to the `RTE` for the originating node.
- Finally, the lifetime of the `RTE` is set to $\max\{t_{\text{old}}, t_{\text{min}}\}$ whereas t_{old} is the existing lifetime and

  $$t_{\text{min}} = t_{\text{current}} + 2 \cdot \texttt{NET\_TRAVERSAL\_TIME} - 2 \cdot \texttt{hopCount} \cdot \texttt{NODE\_TRAVERSAL\_TIME}.$$

  The term $2 \cdot$ `NET_TRAVERSAL_TIME` approximates the worst round trip time of a packet within the net. From this, the approximated fraction of time that already has passed is subtracted. `NODE_TRAVERSAL_TIME` hereby is an estimated delay between two neighboring nodes, including queuing delays, interrupt processing and transfer times.

Now, there are two possibilities: If the node is the requested destination itself or if it knows the destination and the origin node did not set the *destination only* flag (`D`-flag) in the *RouteRequest*, the node generates a *RouteReply*. Otherwise, the node forwards the *RouteRequest*. Thus, setting the `D`-flag in the *RouteRequest* packet disallows replies by nodes other than the destination node itself.

In the first case, the node generates a *RouteReply* packet and unicasts it to the previous hop. This packet is routed along the reverse path to the originating node that was previously created when the *RouteRequest* was forwarded. If the node generating the *RouteReply* is not the destination node itself, it verifies if the *gratuitous* flag (G-flag) is set in the *RouteRequest* packet. If it is set, the node will also unicast a gratuitous *RouteReply* packet towards the destination node in order to ensure that the destination node also has a path towards the originating node (see figure 3.3). A gratuitous *RouteReply* packet looks like a normal *RouteReply* that would have been created if the destination node had sent a *RouteRequest* for the originating node. An originating node should set this flag if the communication is supposed to be bidirectional.

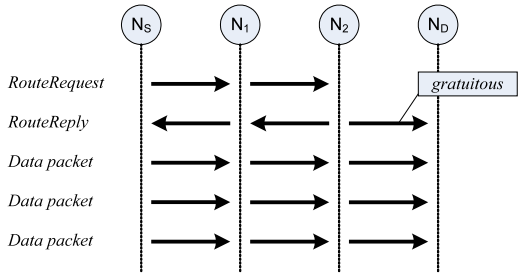


Figure 3.3: This shows an example of a route discovery that is answered by the intermediate node N_2. If the gratuitous flag of the *RouteRequest* is set, it also unicasts a gratuitous *RouteReply* to the destination in order to create a bidirectional route.

In the second case, the node relays the *RouteRequest* packet by broadcasting it if the TTL in the IP header is larger than 1. It modifies the hop count and TTL fields appropriately and sets the destination sequence number field to the maximum of the one received in the *RouteRequest* packet and the one maintained for the destination by itself (if known).

When a node sends or forwards a *RouteReply* packet on a reverse route, it should set the *acknowledgment* flag (A-flag) of the *RouteReply* packet when the link to the current next hop is likely to be erroneous or if there is a danger of a unidirectional link. A node receiving a *RouteReply* packet with such a flag set is supposed to return a *RouteReplyAck* packet to its previous hop. If there is no such packet received, the node assumes a link breakage and proceeds as described in the following section.

3.3.3 Route errors

Whenever a node receives a data packet for a destination for which it has no valid RTE, a *RouteError* packet is generated. Also, when a node detects a link breakage[2]to the next hop of an active route, a *RouteError* is sent if the precursor list of the unreachable neighbor is non-empty.

The *RouteError* is broadcasted if there are more than one precursor for the unreachable destination, otherwise it is unicasted. On reception of a *RouteError* packet, the node checks if it has a valid RTE for the unreachable node. If it has one, it invalidates its entry but only if the next hop to that node is the node from which it received

[2]see section 3.4.1

the *RouteError*. The node forwards the *RouteError* packet if the precursor list in the `RTE` for the unreachable node is non-empty. Again, it is broadcasted if there are more than one precursor and unicasted if there is only one. Thus, the *RouteError* is promoted towards the source of an active route and discarded there.

The *RouteError* may contain more than one unreachable destination. In that case, an intermediate node only forwards these destinations for which it has a non-empty precursor list. Other unreachable destinations are discarded from the *RouteError* packet.

3.4 Connectivity

An important property of mobile ad-hoc networks is the high dynamics of the connections between nodes. Due to node mobility, current connections may get lost, but also new ones are offered. This section describes the mechanisms that are foreseen in AODV to verify node connectivity.

3.4.1 Maintaining local connectivity

A node has to be continuously aware of its connectivity to its neighbors to which it maintains active routes. Otherwise, when the neighbor leaves the radio range, data packets would be transmitted to nowhere and as long as the upper layer does not implement some acknowledgment mechanisms itself, such packet losses would remain undetected.

The RFC 3561 ([PeBRD03]) proposes to use link-layer notifications on one hand. As described in section 2.1.2, the IEEE 802.11 standard uses acknowledgments for unicasted packets as well as a ready-to-send/clear-to-send (RTS/CTS) mechanism to reserve the media for data transmission. When promoting these information to the AODV routing layer, each packet unicasted to the next hop will be acknowledged without any additional traffic by the AODV layer on the radio. So there is no need to use additional features. The major drawback of this method is, that it is specific to the 802.11 protocol and not a common feature of standard MAC protocols. Hence, it won't interact with other MAC protocols and it implies the modification of the 802.11 implementation on the end system to support the promotion of these information.

If link-layer notifications cannot be used, the RFC proposes to use passive acknowledgment. In that case, after a node has sent or forwarded a packet to the next hop, it waits a certain timeout (`NEXT_HOP_WAIT`). If it receives any 'sign of life', i.e. any packet from that node, it assumes the forwarded packet as acknowledged. There is no packet-specific acknowledgment (same as with the *RouteReplyAck* packet), thus, the node anyhow is not sure that a specific packet arrived at the next hop. But still it is in communication range, so the connection is not lost.

To raise the probability of receiving any notification from the neighbor, the node should listen to the medium, requiring it to be in promiscuous mode. A disadvantage of the promiscuous mode is, that it consumes a bit more energy because all packets have to be reassembled on the MAC layer in order to promote them to the higher layer where they also have to be treated. This may be critical to low power nodes like PDAs.

If the node didn't receive any passive acknowledgment within the last `NEXT_HOP_WAIT` time, it is supposed to verify connectivity actively by using a ping mechanism. This may be an ICMP echo request ([Post81]) or an unicasted *RouteRequest* to the neighbor. If it does not receive any reply within `RING_TRAVERSAL_TIME` time, it assumes a link breakage and proceeds as described in section 3.3.3.

3.4.2 Hello messages

To provide connectivity information, an option of the AODV protocol is to broadcast *Hello* messages in case there were no other broadcasts within the last `HELLO_INTERVAL`. A *Hello* message is a *RouteReply* packet with `TTL` set to 1 and the `Destination IP Address` set to the node's IP address; the `Originator IP Address` is irrelevant. The lifetime of the packet is `ALLOWED_HELLO_LOSS` · `HELLO_INTERVAL`.

If a node receives a *Hello* message, it creates a route to that node if necessary and sets its lifetime to at least `ALLOWED_HELLO_LOSS` · `HELLO_INTERVAL`. If a node does not 'hear' anything from a neighbor for which it has an active route within `ALLOWED_HELLO_LOSS` · `HELLO_INTERVAL` time, it assumes a link breakage and proceeds as described in section 3.3.3.

4. The VSR protocol

The Virtual Structure Routing (VSR) is a routing algorithm based on a self-organization mechanism dedicated to hybrid[1] mobile ad-hoc networks. This chapter describes the virtual topology that the routing algorithm relies on and the algorithm itself. Detailed information about the basic functionality can be found in [ThVa04b] and [ThVa05c]. [ThVa04a] and [ThVa05b] are extended works which describe more functionalities.

4.1 Virtual topology

The topology upon which VSR builds up its routes consists of a virtual backbone and clusters (see figure 4.1). The virtual backbone has its equivalent in wired backbones like those of the cellular network GSM. The following subsections first describe the basic ideas, then the actual virtual topology construction.

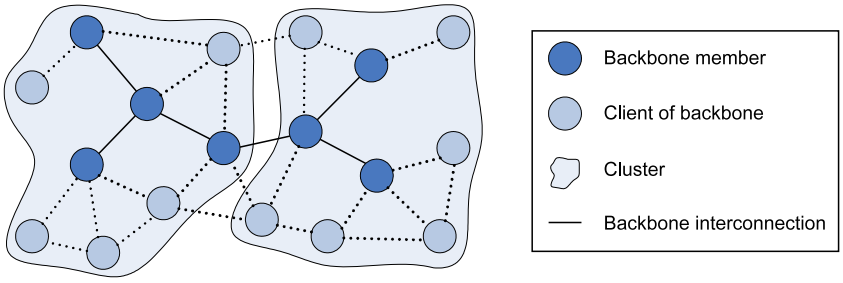


Figure 4.1: Example of a virtual topology used by VSR

4.1.1 Basic ideas

The basic ideas are the construction of a connected set of nodes on one hand, to which the control traffic is concentrated (that is, the construction of a backbone), and the construction of clusters on the other hand, building small, 'handy' and hierarchical sub-topologies.

[1] see section 2.1.1

4.1.1.1 Minimum connected dominating set

A k_{cds} Connected Dominating Set (k_{cds}-CDS) is a set of *dominating* nodes that are connected directly and that are building a backbone upon which the control traffic of the network is concentrated in order to reduce it. Non-dominating nodes are called *dominatees* and have to be in the k_{cds}-neighborhood of a dominating node d:

$$n \in N_{k_{\text{cds}}}(d)$$

whereby $N_{k_{\text{cds}}}(d)$ is the neighborhood of node d that can be reached in k_{cds} hops. Thus, every node in the network can be reached via the backbone. A k_{mcds} Minimum Connected Dominating Set (k_{mcds}-MCDS) is the set M containing the minimum number of nodes of a given network that fulfills this condition. Figure 4.2 shows an example of a 2-MCDS.

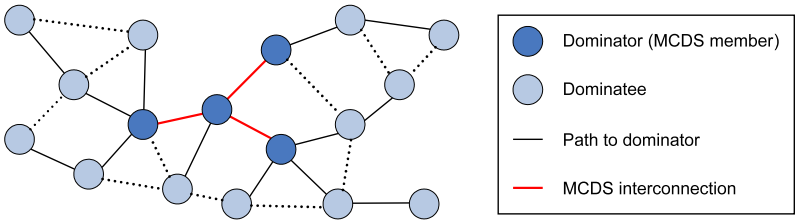


Figure 4.2: Example of a 2-MCDS

Due to the complexity of the construction of a MCDS (NP complete, [ClCJ90]), most algorithms are trying to construct a good MCDS approximation. A construction of a 1-MCDS mostly can be divided into two steps: At first, some dominators are elected, the nodes within their 1-neighborhood becoming their dominatees. This election is similar to the election of a clusterhead (see section 4.1.1.2) and is based on a metric which can be based on the node's ID, the node's degree, energy considerations etc. The second step is to interconnect these dominators. In [CCCD02], the dominatee with the highest number of non-connected neighboring dominators will also become a dominator. If there are nodes with the same number of neighboring dominators, the node with the highest ID wins.

However, these algorithms are not well suited for a k_{mcds}-MCDS construction with $k_{\text{mcds}} > 1$ because they require a high delay that is getting worse with increasing k_{mcds}. They are, however, more interesting to reduce the number of dominators which allows more nodes to save their energy and which simplifies the maintenance of the CDS.

4.1.1.2 Clusters

A cluster in an ad-hoc network consists of a set of nodes that are geographically close. In VSR, every cluster has one clusterhead. Clusters are used to divide the network in service zones. The clusterhead of each cluster serves as a virtual access point providing several services like cluster addressing, localization of nodes and control flow management.

Per definition, to every node n being part of a k_{c}-cluster applies that it has a maximum distance of k_{c} hops to its clusterhead h:

$$n \in N_{k_c}(h)$$

whereby $N_{k_c}(h)$ is the neighborhood of node h that can be reached in k_c hops. In the example figure 4.3, every cluster is a 2-cluster.

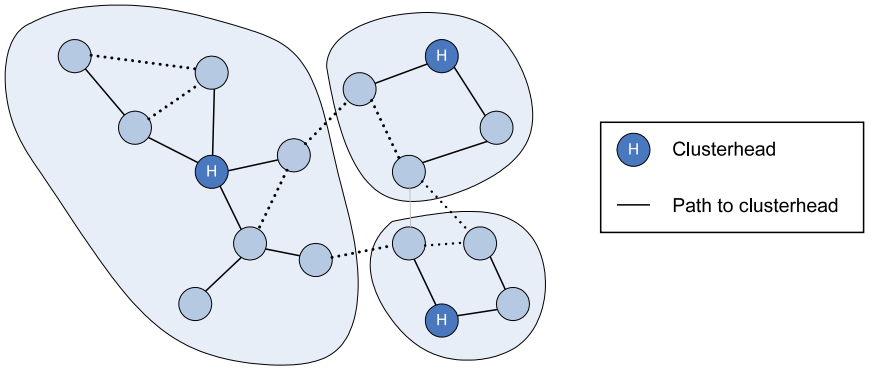


Figure 4.3: Clusters

The aim of the clusterization algorithm is to minimize the number of clusters (thus the number of clusterheads) on one hand, and, on the other hand, to build clusters with a constant diameter. Furthermore, the clusterhead should be in the geographical center of its cluster. For a 1-cluster construction (that is all nodes are in direct communication range to its clusterhead), there are several clusterization algorithms. At the beginning, every node has to learn about its neighborhood and it has to gather some information on it. The information needed depends on the used metric. The simplest metric is based on the identifier, i.e. the address, of a node. The node with the lowest identifier is chosen as the head of the cluster. Other more complex metrics are based on the node's degree (i.e. the number of direct neighbors a node has), the physical distances between nodes, a node's relative mobility, energy considerations and so on.

4.1.2 Construction

In [ThVa04b], a combination of a k_{cds}-MCDS approximation and k_c-clusters is proposed: The CDS builds the actual backbone upon which the control traffic is concentrated and thus reduced because the number of nodes on the backbone is reduced. The clusters represent some sorts of service areas that provide localization and addressing. Also other services such as those of a Foreign Agent in the terms of Mobile IP ([Perk02], [JoPA04]) are imaginable, since this topology may also be connected to an access point providing access to a wired network.

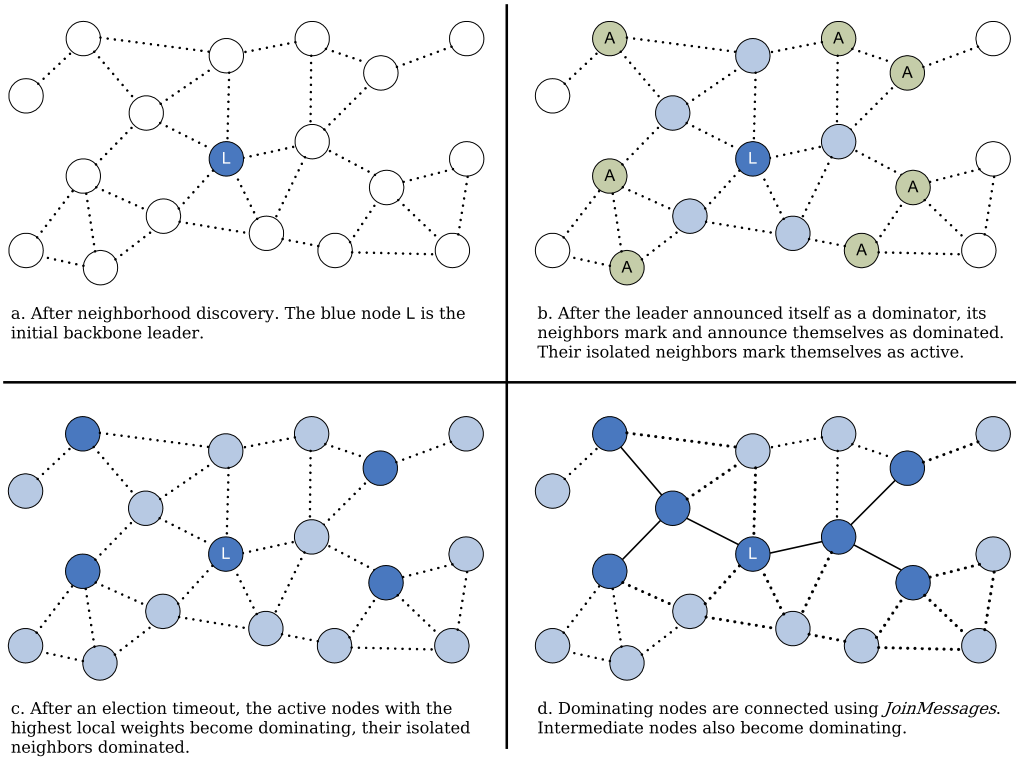
4.1.2.1 The backbone

The backbone of this topology is constructed in several steps. At first, all nodes have to learn about their k_{cds}-neighborhood. This is done via several *Hello* messages that contain a list with all known neighbors within a hop range of k_{cds}. A node floods its k_{cds}-neighborhood with this message, setting its `TTL` to $k_{cds} - 1$. A node then knows all neighboring nodes that can be reached in k_{cds} hops including the next hop node

towards it. Moreover, with these *Hellos*, a node is able to distinguish unidirectional from bidirectional links.

The next step is to construct the backbone, that is, to create a connected subset of nodes. In a hybrid network, the natural leader of the backbone is the access point (AP). In pure ad-hoc networks, the leader is chosen in a distributed way. The role of the backbone leader is solely to initiate the backbone construction, which actually is the construction of a k_{cds}-CDS, and to maintain it[2]. A node can be in one of the following four states:

- `isolated` – This is the starting state of every node, indicating that it is not yet member of the topology. In this state, it just waits for a signal that allows it to join.
- `active` – In this state, a node is in the process for the election as a dominator.
- `dominating` – The node is a dominator, thus being part of the backbone.
- `dominated` – The node is a client of the backbone. Its distance from the backbone is at maximum k_{cds} hops, i.e. the nearest dominator is at most k_{cds} hops far.



a. After neighborhood discovery. The blue node L is the initial backbone leader.

b. After the leader announced itself as a dominator, its neighbors mark and announce themselves as dominated. Their isolated neighbors mark themselves as active.

c. After an election timeout, the active nodes with the highest local weights become dominating, their isolated neighbors dominated.

d. Dominating nodes are connected using *JoinMessages*. Intermediate nodes also become dominating.

Figure 4.4: Construction steps of the virtual topology's backbone used by VSR. The resulting graph (d) is a 1-CDS ($k_{\text{cds}} = 1$).

The backbone leader is per definition the first dominator. It initiates the cascading CDS construction in announcing itself as a dominator through sending a *StateMessage*. The backbone's construction then obeys the following rules (see figure 4.5):

[2]see section 4.1.3

- An `isolated` or `active` node becomes a `dominated` node if it receives a message from a dominator that is less than k_{cds} hops away. It marks the previous hop of that message as its parent node and broadcasts a *StateMessage* announcing itself as a dominatee.
- An `isolated` node that receives a message from a `dominated` node becomes `active` for an election as a dominator and starts an election timeout.
- An `active` node whose election timeout has expired and which has the highest weight[3]among all `active` nodes within its k_{cds}-neighborhood becomes `dominating`.

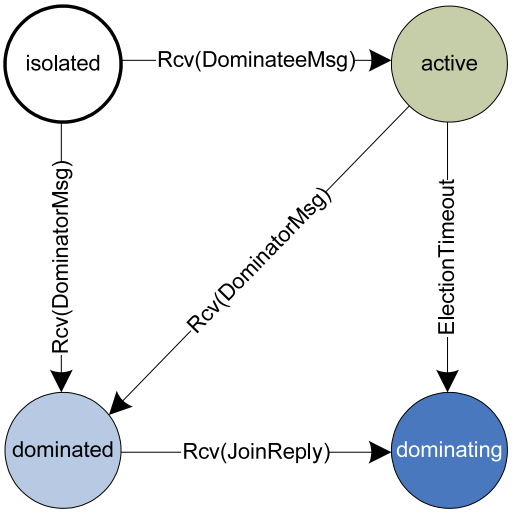


Figure 4.5: State diagram of a node during topology construction

So far, this constructs `dominating` and `dominated` nodes in waves around the leader, propagating towards all nodes until every node is either `dominating` or `dominated` (see figure 4.4, a-c). The third step is to connect these dominators. At the beginning, only the leader is considered to be connected. A connected dominator sends a *JoinMessage* message to invite other non-connected dominators to connect to it and thus allows them to be part of the backbone. The `TTL` of such a packet is set to $2 \cdot k_{\text{cds}} + 1$ and is relayed by all `dominated` nodes.

A non-connected dominator receiving a *JoinMessage* marks itself as connected and sends a *JoinReply* back to the connected dominator on the reverse route. Every dominatee receiving such a packet marks itself as a connected dominator and relays the *JoinReply* message towards the node that previously emitted the *JoinMessage*. The newly connected dominator then emits a *JoinMessage* itself to allow other non-connected dominators to connect. This continues iteratively until all dominators are connected, resulting in a k_{cds}-CDS.

4.1.2.2 Clusterization

Starting from the existing backbone constructed in the previous section, the network is clustered into k_{c}-clusters. Only backbone members participate in the election of a clusterhead, thus reducing overhead.

[3]see section 4.1.4

To create k_c-clusters, dominators have to investigate their backbone neighbors that are at maximum $(k_c - k_{cds})$ hops away. A temporary $(k_c - k_{cds})$-CDS-neighbor table for this is created that will be only used during the cluster construction phase. Additionally, a list that contains all of these nodes, that do not have a clusterhead yet, is maintained. After a timeout τ (message propagation time), the node from this list with the highest weight is chosen as a new clusterhead. After this, all nodes that do not have a clusterhead yet, continue the clusterization process iteratively. In continuation of figure 4.4, figure 4.6 shows the resulting topology after the clusterization process.

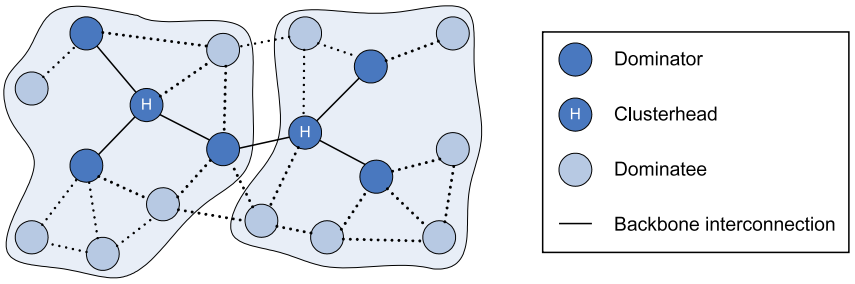


Figure 4.6: Clusterization. This graph shows two 2-clusters ($k_c = 2$) upon the CDS backbone ($k_{cds} = 1$).

4.1.3 Maintenance

Because of node mobility, an important part of such a topology is its maintenance. For the backbone, this means, that all dominatees have to have a dominator and that all dominators have to stay connected.

To allow a fast reaction on topology dynamics, a node has to continuously collect information about its neighbors using *Hello* packets. For the maintenance of the CDS and the clusters, the k_{cds}-neighborhood is relevant. To provide such continuous information, periodic *Hello* messages are emitted by each node.

4.1.3.1 Backbone maintenance

To maintain the connectivity of the CDS, *ApHello* messages are periodically emitted by the leader and relayed along the backbone. So the *ApHello* is propagated on a tree, allowing a dominator to determine its parent/child relationship with other dominators. If the leader is an access point (AP), these messages may contain configuration parameters for Internet access. The absence of l successive *ApHello* messages allows a node to determine whether it has lost its logical parent node towards the backbone leader.

If a dominator d has lost its connection, it checks its neighbor table if there is another dominator nearby which it can choose as an alternative parent node. Such an alternative parent has to be one hop closer to the backbone leader than d was before to avoid circles (the information is propagated within the *ApHello*). If there is one, it simply chooses this node as his new parent. If there is none, it initiates a repair process:

1. The disconnected dominator d broadcasts a *ReconnectRequest* with the sequence number of its last received *ApHello*; its dominatees relay the message

via broadcast. Other dominatees receiving a *ReconnectRequest* message unicast it to its dominator d_i ($d_i \neq d$).

2. A dominating node d_i replies to this message with a *ReconnectReply* if it has received a *ApHello* newer than d. This message is sent via unicast on the reverse route.

3. For every *ReconnectReply* that node d receives, it marks the next hop n_i towards the emitting dominator d_i as a possible *secondary parent*.

4. Finally, node d chooses the node with the highest weight as a new parent among all secondary parents known. For the chosen node n_i, it sends a *ReconnectAdvert* towards the emitter of the *ReconnectReply* d_i; this message forces intermediate dominatees (and therefore also the chosen parent) to become dominating in order to reconstitute the connectivity of the CDS (compare with the *JoinReply* packet, section 4.1.2.1).

If this procedure fails m times, the disconnected dominator d floods its subtree with a *BreakMessage*, forcing all child nodes (and itself) to leave the partitioned subtree and to reinitialize their states to *isolated*. Then, as during construction, these nodes are waiting for an external signal to begin the reconstruction. Connected dominators having such isolated nodes within its k_{cds}-neighborhood send a *JoinMessage* which activates the construction process. As these isolated nodes are not necessarily within the k_{cds}-neighborhood of a dominator, a dominatee has to relay that information to its dominator, thus invoking him to send a *JoinMessage*. An isolated node receiving such a message marks itself as *active* and the construction phase has started.

To reduce the cardinality of the CDS, a dominator continuously verifies that it has at least one dominatee at k_{cds} hops. If it does not have one and if it is a leaf of the backbone (i.e. not having any children on the backbone to which it relays the *ApHello*), it is useless – its dominatees can be served by other dominators. Hence, the node sends a *UselessMessage* and marks itself as a dominatee.

4.1.3.2 Cluster maintenance

The maintenance of the clusters is carried out completely by the backbone members – dominatees do not participate. Within each *Hello*, every node indicates if it is a clusterhead and the hop distance towards its clusterhead (not relevant if it is a clusterhead). With this information, a node maintains its proper clusterhead, remembering the next hop towards it.

If a node has lost its clusterhead, e.g., the next hop towards its clusterhead is gone, it searches in its neighborhood table a dominator that announces a different cluster that is less than k_{c} hops away.

4.1.4 Metric

One aim of VSR's virtual structure is stability. Thus, the main control flow should be limited to only a few nodes building the backbone. To reduce expensive reconstructions of such a structure, the nodes participating at the backbone have to be chosen with care. Furthermore, clusterheads have an even greater importance among

normal dominators as they should maintain a cluster as long as possible. To achieve this, a metric to measure the *weight* of a node is used. The following criteria are taken into account:

1. Persistence

   This forces an important node to effect its role as long as possible. Once a dominator is elected it will remain dominating. Thus, this criteria will not enter into the formula below.

2. Relative mobility

   This favors nodes with a stable neighborhood and since this is the most important impact of mobility, only the neighborhood fluctuation is measured. For this, the ratio between the number of neighborhood changes and the average node degree during the last n time intervals is measured.

3. Energy

   Since energy on small devices is a critical part and since nodes that are important for the structure are likely to consume more energy because of structure maintenance measures, low energy nodes should be penalized heavily in terms of the metric. An exponential metric would be appropriate in this case.

4. Degree

   On one hand, a node's degree (i.e. the number of nodes within communication range) shouldn't be too small in order to reduce the number of dominators and clusterheads. On the other hand, a large number of neighbors increases collision and congestion probabilities. Hence, the metric for the degree assumes an optimal number of neighboring nodes and decreases with increasing and decreasing node numbers.

These parameters normalized, a node's weight is the following:

$$p_{\text{weight}} = \alpha \cdot p_{\text{mobility}} + \ \beta \cdot p_{\text{energy}} + \ \delta \cdot p_{\text{degree}}$$

with $1 \geq \alpha \gg \beta \gg \delta \geq 0$.

4.2 Virtual Structure Routing

As described in section 2.2, there are basically two major classes of routing protocols for MANETs: reactive protocols and proactive protocols. Because the virtual topology described in section 4.1 provides a network structure based on two levels, the routing protocol proposed uses both approaches to its advantage: At the lowest level (i.e. clusters) a proactive intra-cluster routing is considered whereas, at the higher level, a reactive inter-cluster routing over the backbone is proposed. Thus, VSR is a routing protocol completely based on the self-organized topology.

4.2.1 Intra-cluster routing

Routing within a cluster is proactive, that is, every node always has a route to every other node within its cluster. Thus, communications between geographically near nodes have a relative low latency since a route between them is likely to exist. Furthermore, since clusters are less dynamic than the nodes within a cluster, inter-cluster routing over longer distances is much more stable. The induced overhead due to the maintenance of these cluster internal routes can be controlled through the parameter k_c which represents a cluster's radius.

Each member of a cluster periodically broadcasts a *Hello* message that also contains its direct neighbors (its 1-neighbors). Thus, every node is capable to get to know its 2-neighborhood and to determine whether a connection to its direct neighbor is bidirectional. This *Hello* is forwarded by a neighbor only if the link between them is bidirectional, if they have the same clusterhead and if the `TTL` of the message is greater than 1. The initial `TTL` is $2 \cdot k_c + 1$ (the diameter of the cluster). These *Hellos* also contain values like CDS state information that are needed by the k_{cds}-neighborhood for the maintenance procedure of the virtual structure. Therefore, the additional overhead is reduced.

With these *Hello*'s, a node has a global view of its cluster, including bidirectional links between all members. Upon this, a node can apply any source routing algorithm (for example Dijkstra) to find an optimal route to its desired destination node within its cluster. Or it simply can set the node from which it received a relayed *Hello* as the next hop towards the original sender of the node.

4.2.2 Inter-cluster routing

If a node has to send a packet to a destination which is not part of the same cluster and to which it does not have a route yet, a route discovery has to be started. Thereby, as a benefit of the backbone, the number of nodes that are involved in the discovery process is minimized.

Whereas classical routing protocols considered routes as a series of node IDs (or addresses), inter-cluster routing of VSR proposes routes based on a series of cluster IDs. Because clusters are more stable than mobile nodes, routes in VSR appears to be more robust.

4.2.2.1 Cluster topology discovery

As a prerequisite for *cluster routing*, each node of a cluster c_i has to be aware of the local cluster topology, i.e. adjacent clusters of cluster c_i have to be determined. Each node g_i receiving *Hellos* from a node g_j of a different cluster c_j has a gateway functionality from cluster c_i to cluster c_j. This property of a node is propagated towards all members of its cluster with its *Hello* message. Consequently, the length of a *Hello* may increase further. But this is still acceptable, since the number of packets has a greater influence of the performance due to medium access than the length of a packet.

4.2.2.2 Route discovery

For a destination node d_k that is not within the same cluster and that is not within the k_{cds}-neighborhood of the sender s_i, a *route of clusters* is maintained. If s_i does not have a route table entry for d_k yet, it launches the route discovery:

1. If the sender s_i (member of cluster c_i) is a dominatee, it sends its *RouteRequest* directly to its dominator using intra-cluster routing.

2. A dominator generating or receiving a *RouteRequest* and not having the requested destination in its neighborhood table has to forward it on the backbone (backbone flooding). Before it does so, it adds the ID of its cluster to the request if it is different from the last cluster ID marked in the request.

3. A dominator that has the requested node d_k in its neighborhood table acts as proxy and generates the *RouteReply*; it also adds the cluster route towards s_i to its route table. The *RouteReply* is sent back using the inter-cluster routing (a route to s_i already exists because of the previous *RouteRequest*).

4.2.2.3 Routing

Finally, routing is possible after a successful route discovery and applies to *RouteReply* and data packets. The route to the actual destination of such a packet is marked inside the packet in form of a cluster route, that is, a route that only contains the IDs of the clusters to traverse. A node n_j that receives a 'routable' packet processes its route in *reverse* order, starting with the destination d_k. It acts in accordance with the following rules:

1. If n_j has the destination in its neighborhood table, it sends the packet directly towards it using intra-cluster routing.

For every cluster c_m (starting with the cluster of d_k, then using the cluster nearest to d_k, ...):

2. If n_j has a 1-neighbor n_m that is part of the cluster c_m, the packet is forwarded to this node (thus, n_j is a gateway to c_m).

3. If n_j has a 1-neighbor $n_{j'}$ that is a gateway to c_m, the packet is forwarded to $n_{j'}$.

4. If the cluster of n_j has at least one gateway to c_m, n_j chooses the nearest gateway $n_{j'}$ and forwards the packet to it using intra-cluster routing.

If none of these rules can be applied, the next cluster is tried until it comes to a known cluster where the algorithm finally stops.

Since all nodes of a cluster generally have the same complete view of their cluster's topology they will make coherent decisions and the packet will come closer towards its destination by every hop. However, slight incoherencies may be encountered due to topology changes and propagation delays of these changes. Thus, to avoid loops, a packet will be discarded silently if it is received a second time. It is identified by the couple (source address, sequence number).

Every node that is supposed to forward a packet tries to send it to the cluster nearest to the destination. Thus, the route of a packet is dynamic and a node that is modifying the route has to update the route marked inside the packet so that the destination can benefit directly from the new route.

4.2.2.4 Acknowledgment and route repair

To increase the delivery ratio, an acknowledgment mechanism is used when forwarding a data packet. Similar to the proposition in the RFC of AODV as described in section 3.4.1, [ThVa05b] proposes the use of the 802.11 MAC layer acknowledgment facility. Besides this, a forwarding node may listen to the medium in promiscuous mode. If it 'hears' the next hop forwarding the particular packet, it takes this as a passive acknowledgment. Solely the destination itself has to send an active acknowledgment as it is not supposed to forward the packet.

If there is no acknowledgment after a timeout t, the packet is retransmitted. After n_{retry} retransmission attempts, a route repair is initiated. For the route repair, the routing algorithm described in section 4.2.2.3 is re-executed. But this time, the erroneous node is considered to be dead and thus not taken into account.

5. Implementation

This chapter is dedicated to the implementation of the AODV protocol described in chapter 3. We chose to implement it in the simulation environment from 'scratch' (that is, from the RFC) for several reasons: It did not exist in the simulator's version used, we wanted to have an implementation as compliant to the RFC as possible and we wanted to be sure to use the same mobility and traffic generation model implementations that were used by VSR. Furthermore, the types of statistics collected should be similar.

5.1 OPNET Modeler

As simulation environment, OPNET Modeler Version 8.1.A was used. OPNET Modeler is an event-driven network simulator tool that supports many kinds of networks, from office LANs, WANs, Wireless LANs, telecommunication networks up to satellite networks.

OPNET Modeler models a network with several layers. The highest layer is a global network (world network) that may contain several sub-networks and their linkage. A network may also contain nodes which are represented by node models. A node model describes the interface of a node as well as its configuration parameters and its behavior. An interface may be any kind and number of communication interfaces (Ethernet wire, IEEE 802.11 radio etc.). It also describes the mobility of a node (mobile or fixed). Furthermore, the supported protocol stack is defined within a node model. Every protocol and its interfaces with higher and lower layer protocols are described using process models. As example, figure 5.1 shows the AODV node model implementation used for these simulations.

Process models are described using a state transition diagram of a finite state machine (FSM). There are *unforced* and *forced* states: An unforced state represents a persistent change in the condition of a process, thus the process will remain in that state until there is another event causing a state transition. A forced state will return immediately after executing the code associated with it, thus no simulation time will elapse in this state. Since AODV is a stateless protocol, it only has one unforced state – all others are forced (not counting initial wait states which are clearly not part of the AODV protocol; see figure 5.2).

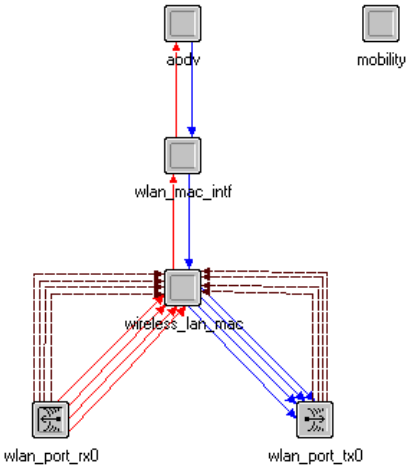


Figure 5.1: Node model for the AODV implementation. The upper four gray boxes are placeholders for process models, the arrows between them are communication channels between the processes. The lower two boxes are the actual physical link layer interfaces.

State transitions may be unconditional (the transition will always fire) or may only be taken if a condition is fulfilled. The latter one may be any boolean expression and often is a test for a specific interrupt type like an incoming packet or a timer interrupt. Additionally, a function that is called whenever that transition is taken may be defined.

The actual user source code (in C) only describes the detailed behavior of the process. Pieces of code may be associated to the enter executive of a state, to the exit executive of a state and to each state transition. To facilitate reuse of code, functions may be defined separately.

Thus, OPNET Modeler follows a real modeling approach with a simple and clear graphical representation. This minimizes implementation errors especially for stateful protocols and avoids undesired cross-layer references due to a separate definition of the interface of a node or a process.

5.2 Implementation particularities

OPNET Modeler provides powerful statistic collection and analyzation methods as well as different traffic generation and mobility models. For the evaluation and comparison of AODV and VSR, they were not used. Instead, statistic collection, traffic generation and mobility models were implemented separately. This decision was made to assure the desired traffic and mobility models and to avoid an even slight distortion of the statistics due to OPNET particularities.

Furthermore, to observe the protocol's true behavior without any overhead added through the IP implementation of OPNET, AODV and VSR were implemented directly on top of the 802.11 MAC layer (compare with figure 5.1). Some features of the IP protocol like the time to live (`TTL`) field of the IP header used by AODV were added directly to the AODV protocol.

As already mentioned, AODV is a stateless protocol, thus it was implemented using only one unforced state (`idle` state) and several forced states that are rather event

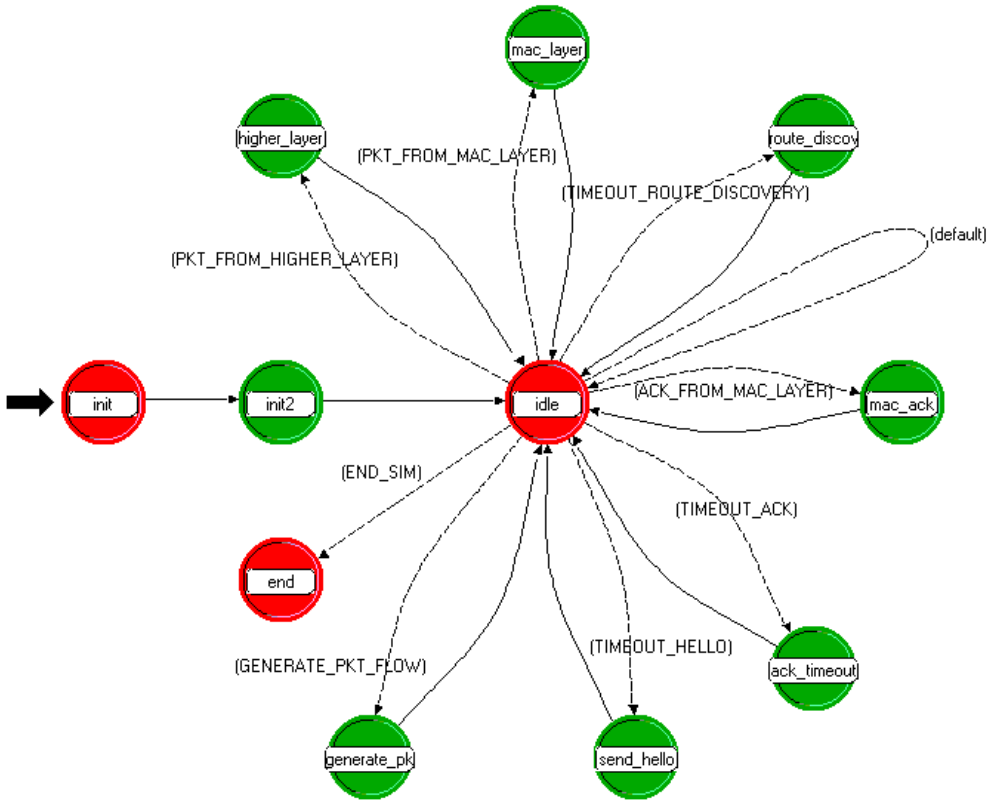


Figure 5.2: Process model for the AODV implementation. Red states are unforced states, green states are forced states. The dotted transitions are conditional whereas the solid ones are unconditional. The upper part of each state 'contains' the entry executive and the lower part the exit executive. For forced states, the latter is superfluous.

handlers than states (figure 5.2). The purpose of the initial unforced `init` state is to wait for the initialization of the MAC protocol. The forced state `init2` loads the configuration parameters and schedules initial events for traffic generation. The unforced `end` state is used to release resources at the end of the simulation. The other states have the following purposes:

- `generate_pk_flow`

  In this state, a node schedules all events for all packets that are supposed to be sent as part of a flow (events `PKT_FROM_HIGHER_LAYER`). Furthermore, an event for the node is scheduled, that is supposed to be the source of the next data flow (event `GENERATE_PKT_FLOW`). This node is chosen randomly.

- `higher_layer`

  Here, packets coming from the higher layer, i.e. transport or application layer, are processed. If there is already a route to the destination node, the packet is sent directly. Otherwise, a route discovery cycle is initiated.

- `mac_layer`

  Packets that are passed from the MAC layer to the routing layer are handled here. Depending on the packet's type, the appropriate action is executed.

- `route_discovery`

  This state implements the route discovery cycle including expanding ring search. For each step during the expanding ring search and for each discovery retry, it schedules `TIMEOUT_ROUTE_DISCOVERY` until a route is found or until the maximum number of retries has exceeded.

- `mac_ack`

  If the option '`node.Use WLAN-MAC Acks/NAcks`' is enabled, layer 2 notifications for packet acknowledgment are handled here.

- `ack_timeout`

  When sending or relaying a data packet, the node waits a certain timeout for a passive acknowledgment. If this timeout exceeded without an acknowledgment, the node tests the connectivity to the next hop actively using a ping mechanism; this is done here.

- `send_hello`

  If the option '`node.aodv.Hello Messages`' is enabled, a node periodically broadcasts *Hello* messages. In this state, the node checks whether it has broadcasted any messages during the last `HELLO_INTERVAL` time. If it hasn't broadcasted any message, it broadcasts a *Hello*.

5.3 Traffic generation

The traffic generation model is implemented directly within the process models of the protocols. This speeds up simulation time slightly since there is no need to package any interface control information (ICI) and to send it via a vertical cross layer communication channel. The tradeoff of this method is, that it is part of the protocol's process model, thus not clearly separating protocol behavior and traffic generation, even though the code of the traffic generation is limited to its specific forced states (see figure 5.2).

The traffic simulated consists of one-way streams with a constant bit rate (CBR). The source of the stream as well as the destination are chosen randomly with a uniform distribution. The traffic generation model takes four simulation parameters:

- the size of a packet in bytes
- the time between two packets of a flow (packet interarrival time)
- the number of packets per flow
- the number of simultaneous flows at the same time

Thus, the duration of a flow is packet interarrival time $\times$ packets per flow.

5.4 Mobility model

The process that controls the mobility of a node is defined within the node model as a separate processor. The update of a node's position is made in a discrete manner every `MVT_STEP` seconds. Several mobility models were implemented:

- Random direction

  A direction is chosen randomly with a uniform distribution. The node will continue to move into that direction until it reaches the simulation area's boundaries, then it choses a new random direction.

- Boundless

  On every movement step, the node choses a new direction. When it reaches the simulation area's boundaries, it reappears on the other side of the area. The idea is to realise some sort of a spherical area, but, of course, it is not really a sphere, since the radio waves don't follow this behavior. In fact, if a node reappears on the other side, it suddenly vanishes from it's communication context. Only if its neighboring nodes follow the same direction, this context is preserved.

- City

  The node choses a target coordinate and moves towards it on a grid. It first moves in the x direction until it reaches the target's x-value, then in the y direction. This should emulate the movement on streets within an urban structure.

- Random waypoint

  The node first choses a target coordinate and then moves towards it. When it has reached its destination, it chooses a new target.

From these models, the random waypoint model is the most realistic one and widely accepted. Hence, it was the only one used when comparing AODV with VSR.

5.5 Simulation parameters

There are three different simulation series, all of them supposed to compare the comportment of the protocols against scalar properties, and not against the time:

1. Number of nodes in the network

   This series varies the number of participating nodes in the network. Additionally, the simulation area's size is adjusted to have the same node degree and to avoid disconnectivity when only using few nodes. The associated configuration file is `op_models_aodv/_aodv_nodes.ef`.

2. Mobility of the network

   Here, node speed is varied. All nodes move with the same speed. This series uses `op_models_aodv/_aodv_mobility.ef` as configuration file.

3. Number of simultaneous connections

   Finally, with this series, the impact of a traffic augmentation is tested. It takes `op_models_aodv/_aodv_connections.ef` as configuration file.

This section describes the parameters that may be adjusted within these configuration files, sorted by category.

Mobility parameters:

- '`node.mobility.Movement step`' (refers to `MVT_STEP`)

  Time interval after which the node positions have to be updated (in seconds).

- '`node.mobility.Speed_Class`'

  Specifies which speed class to use. Possible values are `NO_MOBILITY`, `LOW_MOBILITY` and `HIGH_MOBILITY`.

- '`SP_LOW_MIN`' and '`SP_LOW_MAX`'

  Defines the interval of the speed value for the low mobility class in meters/second.

- '`SP_HIGH_MIN`' and '`SP_HIGH_MAX`'

  Defines the interval of the speed value for the high mobility class in meters/second.

- '`SP_LOW_MOBILITY_MODEL`' and '`SP_HIGH_MOBILITY_MODEL`'

  Specifies the mobility model to use for the respective speed class. Possible values are:

  0 – no mobility
  1 – random waypoint model
  2 – random direction model
  3 – boundless model
  4 – city model

  See previous section 5.4 for a description of these models.

- '`node.Trajectory: Create file`'

  During simulation, if enabled, produce a trace file of the trajectory used. The format of this file conforms to OPNET's format of a segment-based trajectory file with a fixed time interval (`MVT_STEP`). This option must not be enabled if '`Trajectory: Use file`' is set to enabled.

- '`node.Trajectory: Use file`'

  If enabled, don't use any mobility models, but use a trajectory file (for example one, that was previously created with the '`Trajectory: Create file`' option). This option must not be enabled if '`Trajectory: Create file`' is set to enabled.

Parameters for the traffic generation model (see also section 5.3):

- 'Data Flow: packet size'

  Size of one data packet in bytes.

- 'Data Flow: packet interarrival time'

  Time between two packets of the same flow in seconds.

- 'Data Flow: packets per flow'

  Number of data packets to send for one flow.

- 'Data Flow: number of flows'

  Number of simultaneous data flows in the network.

Parameters for AODV:

- 'node.Maintain local connectivity'

  Apply actions to maintain local connectivity as described in section 3.4. If this option is enabled, one of the following three options should be used.

- 'node.Use WLAN Promiscuous Mode'

  Listen to the medium in promiscuous mode to determine local connectivity.

- 'node.Use WLAN-MAC Acks/NAcks'

  Propagate WLAN-MAC acknowledgments to the AODV layer and use them as acknowledgments for neighbors.

- 'node.aodv.Hello Messages'

  Send hello messages if there is an active route.

6. Evaluation

In [ThVa05b], the VSR protocol was compared with another routing protocol for MANETs, the Cluster Based Routing Protocol (CBRP, [JiLT99]). The CBRP also uses clusters to face the high dynamics due to node mobility. But unlike VSR, there is no backbone to collect the control traffic. Furthermore, CBRP allows to use unidirectional links, which on one hand can exploit otherwise unused links, but on the other hand disallows any acknowledgment and therefore retransmission mechanisms which can have a great influence on reliability.

The goal of this study is to compare the performance of a flat routing protocol (AODV) with a self-organized routing protocol (VSR). This time, both approaches are completely different: VSR with a complete virtual structure that has to be maintained besides routing and data transfer, and AODV with a dynamic, on-demand route discovery and no structure at all. Obviously, this results in some incomparable properties: VSR, on one hand, shows a latency at the beginning, the *topology construction time*. AODV, on the other hand, has a latency for each route request, the *route construction time*. The evaluation shown here only considers the performances during run time, that is, the topology construction time was not taken into account (for a convergence evaluation of the virtual topology see [ThVa05a] and [ThVa04b]). In contrast, the latency of AODV's route construction was included because it influences the overall delivery delay.

6.1 Configuration

This section describes the simulated environment and the protocol specific parameters used within this environment.

6.1.1 General settings

The parameters of the simulated mobile ad-hoc network were the same for both protocols:

1. Radio link

The network was modeled with the IEEE 802.11b link layer and medium access protocol. The physical radio range was 300 m and also equal for all nodes. Thus, all links were bidirectional.

2. Mobility

   As movement model, the random waypoint model as described in section 5.4 was used. The constant speed of every node was 5 m/s. When evaluating the impact of mobility, the speed is varied but always the same for every node.

3. Node density

   The node density always was kept equal. Generally, there are 40 nodes on a squared area with a length of one edge of 2 100 m. When evaluating the scalability of the protocols, the number of nodes in the network is varied. But at the same time, the size of the simulated area is adjusted to avoid network partitioning.

4. Traffic

   The traffic was simulated with 3 concurrent data flows. The source and the destination nodes were chosen randomly for each flow. One flow consists of 8 data packets with a size of 128 bytes sent every 0.25 seconds. This gives a bitrate of 4 kbit/s and a flow duration of 2 seconds. Thus, when talking about network load here, we do not mean the data throughput but the number of clients using the network for data transmission. When evaluating the influence of the network load, the number of flows is increased.

5. Layer 2 notifications

   For genericity and for the reasons mentioned in section 3.4.1, the acknowledgment features of the 802.11 MAC layer are not used for both protocols, although this likely would improve protocol performance. The promiscuous mode was used neither – this allows a node to save energy since frames will be dropped on a very low layer unless they are broadcast or destined for the current node.

6.1.2 AODV configuration

The maintenance of local connectivity in AODV was enabled because otherwise, a node is not able to determine whether the next hop of an active route is still within radio range. As link layer notifications are not used and as there is no active acknowledgment for data packets, passive acknowledgments are the only way to monitor local connectivity. The probability to receive such passive acknowledgments is additionally reduced since promiscuous mode was disabled.

For this reason, the optional *Hello* messages of AODV were enabled. This increases the number of control packets that can be taken as a passive acknowledgment. Otherwise, disabled *Hellos* would have led to a ping and a ping-reply (represented as unicast 1-hop *RouteRequests* and appropriate *RouteReplies*) for every data packet.

RouteReplyAcks were not requested at any time. Since this mechanism is supposed to detect unidirectional links, this was not necessary. All nodes have the same radio range, thus bidirectional linkage is available.

RouteReplies from intermediate nodes were permitted (`destination only` flag of a *RouteRequest* was set to `FALSE`). But since AODV generally creates bidirectional routes with only one discovery process, gratuitous *RouteReplies* were requested (see section 3.3.1).

Since expanding ring search is an approved technique to reduced unnecessary network wide flooding, it was enabled.

All other parameters are using the default values as proposed in the RFC ([PeBRD03]).

6.1.3 VSR configuration

The main configuration parameters of VSR are the maximum distance of a node to the backbone (k_{cds}) and the maximum cluster radius (k_{c}). Increasing the values of k_{cds} and k_{c} leads to more overhead, decreasing them leads to a higher node participation in the backbone resulting in a higher energy consumption. A good compromise is considered to be $k_{\text{cds}} = 2$ and $k_{\text{c}} = 3$ ([ThVa04a], [ThVa05b]).

Even though VSR supports a hybrid structure, that is, nodes communicating in ad-hoc mode with other nodes and/or access points, it may also run in pure ad-hoc mode. Thus, for the comparison with AODV, no access points were used.

The three different *Hellos* are sent in the following intervals: *ApHellos* (generated by the leader and propagated on a tree towards each node) and *ClusterHellos* (generated by each clusterhead and propagated towards each cluster member) are sent every 2 seconds, general node *Hellos* (containing 2-neighborhood and state information) every 4 seconds.

6.2 Criteria

Since the convergence of VSR's virtual topology is not considered here (see [ThVa05a] instead) and since AODV has no convergence delays at all, no evaluation against simulation time was made. In fact, the comparison against scalar attributes is more interesting because it is more generic and independent of elapsed simulation time.

The three most important scalar attributes are considered to be node mobility, the number of nodes in the network and the traffic in the network. In all three cases, at least delivery ratio and delivery delay were compared between the protocols.

6.2.1 Mobility

In this section, the influence of the node mobility is regarded. All parameters remained as described in section 6.1, except for the node speed that is varied between 0 m/s and 30 m/s.

The obtained delivery ratios for VSR and AODV is shown in figure 6.1. Since the routes of VSR are based on a list of clusters rather than on a list of nodes, the VSR routes remain much more stable. The dynamic routing through the clusters can lead to an adapted route for each data packet. In AODV, however, high mobility leads to *RouteErrors* requiring an expensive reconstruction of the route. Moreover, the data packet causing the *RouteError* is dropped.

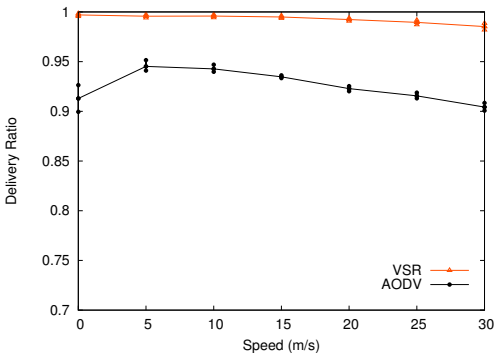


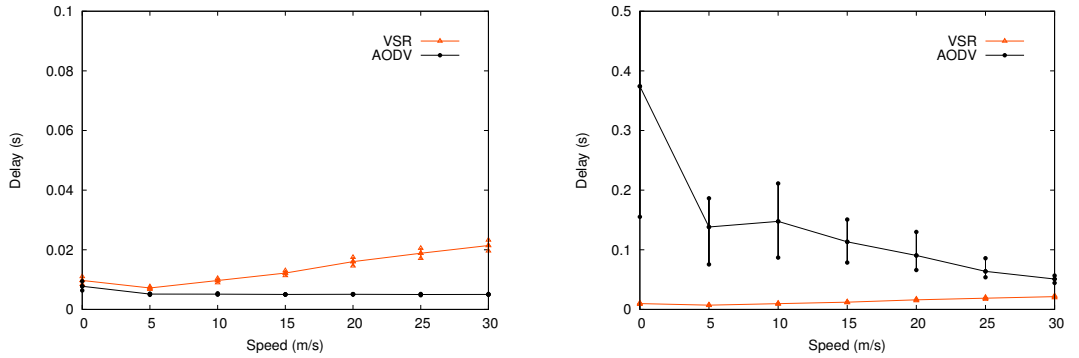
Figure 6.1: Delivery ratio of data packets versus node speed

The slump of AODV's delivery ratio in a static network (no mobility) may have been caused by the use of passive acknowledgments (i.e. no packet specific acknowledgment). Even if single data packets are lost, a node may still 'hear' something from the next hop. This may happen when there are many nodes in a small area and frequent collisions occur. If the topology is changing over time, such situations only have a temporary character. In a static network, a node may always rechoose the same node with a bad link as a next hop. Using active acknowledgment with a retransmission mechanism (like in VSR) can avoid this problem.

In figure 6.2(a), the delay between the emission of a data packet by the originator and its reception by the respective destination is shown. The amount of network mobility slightly influences the performance of VSR, while with AODV, delay remains constant. AODV's routes are always fresher and therefore more optimal because they have a shorter lifetime. VSR routes adapt to topology changes slightly slower which results in the effect that they are not always optimal, thus have a higher delay.

At a first glance, this might be an advantage for AODV, but at a second thought, it can be explained by the fact that undelivered packets are not taken into account (see delivery ratio above). If there is an error in relaying a packet towards the next hop, AODV ignores it and drops the packet, whereas VSR attempts a retransmission which costs time.

Additionally, the real end-to-end delay is the delay experienced by the user, that is, the latency of the route construction time needed by AODV has to be considered. Figure 6.2(b) shows, that the on-demand routing of AODV has a heavy impact on the delay between packet generation and reception. VSR, on the other hand, benefits from its already existing intra-cluster routes. However, the tested scenarios penalize on-demand routing a bit since the simulated traffic consists of short unidirectional streams. Bidirectional and longer streams could have been a slight advantage to AODV since it always constructs bidirectional routes with one request. Furthermore, the short traffic flows used may also explain the strange behavior of AODV's end-to-end delay that *decreases* with increasing speed: If a route construction or reconstruction fails due to high mobility, the remaining packets of that flow do not influence the delay since they never reach their destination. The impact on the delivery ratio, however, remains moderate since there are only few packets per flow in the tested scenarios.



(a) Delay between *emission* and reception (b) Delay between *generation* and reception

Figure 6.2: Delay of a data packet versus node speed

Concerning the average route length, both protocols are nearly equally good (see figure 6.3). With increasing speed, VSR tends to slightly longer routes. This is caused by the proactive intra-cluster routing that leads to the effect that the routes are adapted a bit slower. But the dynamic cluster address based routing algorithm faces this sufficiently. AODV, in this case, benefits of the short streams because the routes have a short life time due to high mobility which fits good into AODV's short expiration time of unused routes.

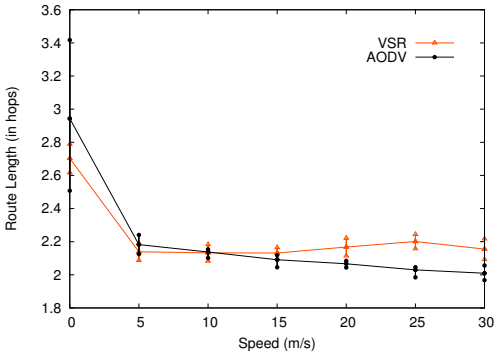


Figure 6.3: Average route length versus node speed

6.2.2 Network size

Network size here means the scalability regarding the number of nodes, that is, measuring the influence of the number of nodes participating in the network on the delivery ratio and delivery delay. With an increasing number of nodes, the simulated area was increased to keep the average node density and the average node degree (i.e. number of neighboring nodes) constant.

Figure 6.4 shows that in VSR an increasing number of nodes hardly has an influence on the delivery ratio. The cluster size and thus the local conditions remain constant. With an increasing number of nodes the number of clusters increases. Accordingly, the size of the backbone increases, but much more slowly. Thus, VSR scales efficiently with the number of nodes.

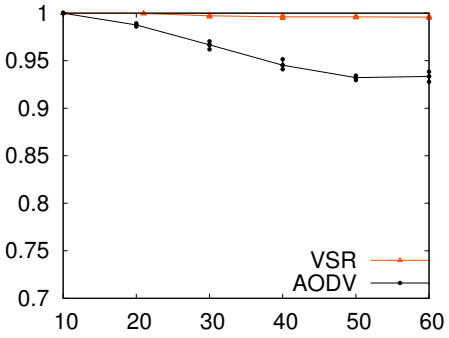
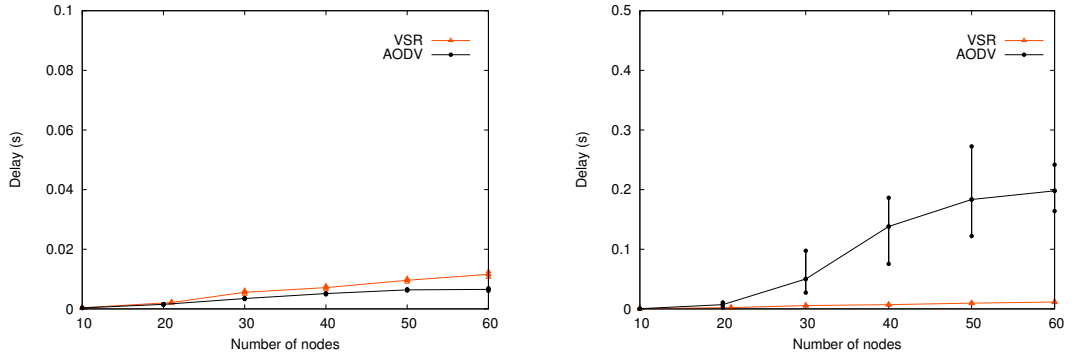


Figure 6.4: Delivery ratio of data packets versus number of nodes in the network



(a) Delay between *emission* and reception

(b) Delay between *generation* and reception

Figure 6.5: Delay of a data packet versus number of nodes in the network

The delivery ratio of AODV, however, is affected by the increasing number of nodes and the growing simulation area. Although, local conditions remain also the same, the average route length is increased. But with every additional hop, there is a risk of a linkbreakage to the next hop. Since there is no packet specific acknowledgment and since AODV does not integrate any retransmissions, the probability of a packet loss increases.

For the same reason as explained in section 6.2.1, the delay between emission and reception of a data packet is slightly better for AODV, but negligible (figure 6.5(a)). The average delay increases with a growing network, which is obvious, since the average route length increases (i.e. the number of hops increases).

With a growing network, AODV needs more time for route construction (figure 6.5(b)). This can be explained with the expanding ring search mechanism used during route discovery. Additionally, between each cycle of the expanding ring search, the waiting time before the next cycle is started increases (see section 3.3.1). But this latency does not increase endlessly: AODV only tests 4 rings (1 hop, 3, 5 and 7 hops) before it floods the entire network. VSR, however, always floods its complete backbone for inter-cluster route discovery, thus, the cluster route construction time increases much more slowly with an increasing number of nodes.

6.2.3 Traffic

In this section, the scalability regarding network load is compared between the two protocols. During the last two sections, the default traffic consisted of 3 unidirectional data flows[1]. Now, different numbers of flows were simulated, from 1 flow to 7 simultaneous flows.

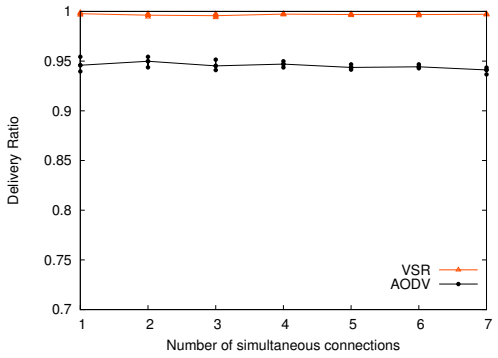


Figure 6.6: Delivery ratio of data packets versus number of simultaneous connections

Considering the delivery ratio, both protocols were stable and the tested network load hardly influenced the delivery. For AODV, the delivery was at about 94 %. This could be expected because of prior results with default values during mobility evaluation (figure 6.1, speed of 5 m/s) and during scalablitiy towards network size evaluation (figure 6.4, 40 nodes).

When comparing the delay between emission and reception of a data packet (figure 6.7(a)), one can see that it still remains constant, thus, the tested traffic was still moderate so that it did not result in too much collisions and congestions.

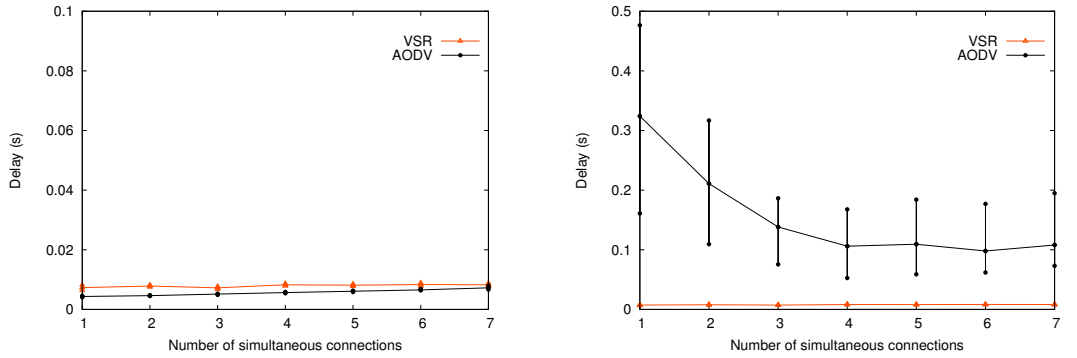
Looking at the delay that comprises the latency of the route construction (figure 6.7(b)), AODV benefits of higher traffic. If there is a higher load within the network, there are more existing routes. Thus, if a node sends a *RouteRequest*, there is a higher chance that an intermediate node already knows the requested destination and that it replies to the originator to speed up route construction. It does, however, not reach VSR's delay that profits from its existing intra-cluster routes.

6.2.4 Control flow

Measuring the overhead is critical for protocols that have to construct and maintain a topology, even if there is no traffic at all. Additionally, for on-demand routing protocols there are a number of route requests each time a new data packet has to be transmitted. Table 6.1 is a sample of VSR's and AODV's control flow. It was measured in control packets per node per second and not in kbit/s since all control packets are considered to be small in size and thus the number of medium accesses is more important than the time during which the medium is occupied.

The first two rows show the overhead of both protocols with the very default simulation parameters. In total, AODV produces about 56 % less overhead than VSR which is related to the absence of a topology that has to be maintained. In AODV, there

[1] see section 6.1.1, item 4



(a) Delay between *emission* and reception (b) Delay between *generation* and reception

Figure 6.7: Delay of a data packet versus versus number of simultaneous connections

	# Flows	Hellos	Topology	RREQ	RREP	RERR	Acks	Total
VSR	3	3.90	0.21	0.005	0.003	–	0.58	4.70
AODV		0.58	–	0.614	0.877	0.008	–	2.08
VSR	7	3.90	0.42	0.010	0.005	–	1.34	5.67
AODV		0.29	–	1.399	2.022	0.022	–	3.73

Table 6.1: Control flow in number of packets per node per second

are about 43 % more *RouteReplies* than *RouteRequests*. This is due to the fact, that intermediate nodes replying to a *RouteRequest* will send a gratuitous *RouteReply* to the requested destination and, additionally, there may be several nodes replying to that request.

The last two rows show the same measurements for the highest network load simulated (7 simultaneous data flows). For AODV, the number of *RouteRequests* and *RouteReplies* almost grow linearly with the number of flows. The number of *Hellos*, however, decreases. This is related to the fact that AODV's *Hellos* are emitted only conditionally, that is, only when there was no other broadcast made by the node. With an increasing number of *RouteRequests*, however, there are more broadcasts, thus, less *Hellos* are needed.

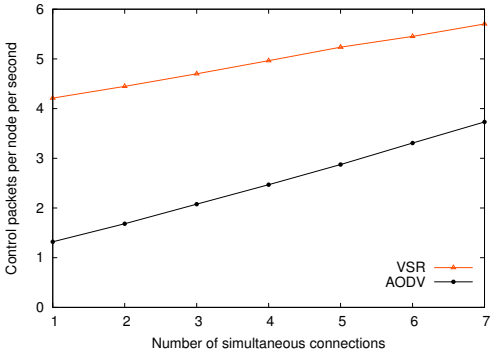


Figure 6.8: Total control flow versus number of simultaneous connections

Figure 6.8 shows the overall overhead added by each protocol. The control flow grows almost linearly for both protocols, but it increases slightly slower in VSR. In

AODV, the increasing number of *RouteRequests* and *RouteReplies* are responsible for that, in VSR, the acknowledgment packets cause the augmentation; but even with the highest traffic tested, AODV does not reach the overhead that was produced by VSR. Thus, AODV produces much less overhead, which was expected due to the different nature of the protocols.

7. Conclusion and future work

Computing nowadays is not only portable as the early notebooks but has become truly mobile and can be found in handheld organizers like PDAs, SmartPhones and even smaller devices. But building up an infrastructure to allow such devices to connect to a network for synchronization and communication purposes is expensive and not always possible. Therefore, direct wireless communication that can be set up spontaneously and without the need of any interconnection equipment is an highly interesting subject. Furthermore, since such devices become more and more widespread, connecting several of them to a complete mobile ad-hoc network allows one to reach more devices that are far beyond one's radio range.

A crucial thing in such networks is routing since it often has to happen over multiple hops, each of them showing an own dynamic and differences in properties like available power, memory and radio range. The number of proposed routing protocols shows that there are many different approaches. The aim of this work was to compare two different ones: a simple light-weight reactive protocol (AODV) that already achieved acceptance and that was standardized by the Internet Engineering Task Force (IETF) with a new hybrid complex approach based on a self-organization and which finally combines both, proactive and reactive mechanisms (VSR).

The main differences between AODV and VSR lie in reliability, latency and produced control flow. AODV does not implement any packet specific acknowledgments which results in a lower delivery ratio since lost or destroyed packets are not retransmitted by the routing layer. VSR, however, adds an overhead due to control flow for topology maintenance but which increases only slowly with increasing traffic or an increasing number of participating nodes. Thus, for a large network with a high usage, VSR is better suited than AODV, since it has lower latency and higher reliability. The fact, that VSR also integrates access points to allow access to a LAN and therefore possibly to the Internet, makes it even more interesting for campus networks and networks on exhibitions and trade fairs. Since such networks also have the property that they are geographically fixed, VSR's clusters may remain very stable.

However, some optional features of AODV like local repair were not tested here. Furthermore, the traffic simulated only tested the protocols' behaviors regarding

the number of clients using the network. The behaviors regarding data throughput still have to be compared. Finally, routing protocols of different types have to be compared in a real environment. While implementations of AODV already exist for several platforms, VSR has still to be implemented into a real IP stack of an existing operating system.

Bibliography

[CCCD02] M. Cardei, X. Cheng, X. Cheng and D.-Z. Du. Connected domination in ad hoc wireless networks. Sixth International Conference of Computer Science and Informatics (CSI), North Carolina, USA, March 2002.

[CJLM+01] T. Clausen, P. Jacquet, A. Laouiti, P. Mühlethaler, A. Qayyum and L. Viennot. Optimized Link State Routing Protocol for Ad Hoc Networks. Fifth IEEE International Multi-Topic Conference (INMIC), Lahore, Pakistan, December 2001.

[ClCJ90] B. N. Clark, C. J. Colbourn and D. S. Johnson. Unit disk graphs. *Discrete Mathematics* Band 86, 1990, pp. 165–177.

[JiLT99] M. Jiang, J. Li and Y. C. Tay. Cluster Based Routing Protocol (CBRP). Internet Draft (draft-ietf-manet-cbrp-spec-01.txt), IETF, August 1999.

[JoMH04] D. B. Johnson, D. A. Maltz and Y.-C. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Internet Draft (draft-ietf-manet-dsr-10.txt), IETF, July 2004.

[JoPA04] D. Johnson, C. Perkins and J. Arkko. Mobility Support in IPv6. RFC 3775, IETF, June 2004.

[OgTL04] R. Ogier, F. Templin and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684 (Experimental), IETF, February 2004.

[PeBRD03] C. Perkins, E. Belding-Royer and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), IETF, July 2003.

[Perk02] C. Perkins. IP Mobility Support for IPv4. RFC 3344, IETF, August 2002.

[Post81] J. Postel. Internet Control Message Protocol (ICMP). RFC 792, IETF, September 1981.

[RoPe00] E. Royer and C. Perkins. Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing. Internet Draft (draft-ietf-manet-maodv-00.txt), IETF, July 2000.

[Schi03] J. Schiller. *Mobilkommunikation.* Pearson Studium. 2nd edition, 2003.

[Tane03] A. S. Tanenbaum. *Computer Networks.* Prentice Hall International. 4th edition, 2003.

[ThVa03] F. Theoleyre and F. Valois. Topologie Virtuelle pour Réseaux Hybrides. Research report, Institute National de Recherche en Informatique et en Automatique (INRIA) Rhône-Alpes, December 2003.

[ThVa04a] F. Theoleyre and F. Valois. Robustness and reliability for virtual topologies in wireless multihop access networks. Mediterranean Ad Hoc Networking Workshop (MedHocNet), Bodrum, Turkey, June 2004.

[ThVa04b] F. Theoleyre and F. Valois. A virtual structure for mobility management in hybrid networks. IEEE Wireless Communications and Networking Conference (WCNC), Atlanta, USA, March 2004.

[ThVa05a] F. Theoleyre and F. Valois. About the self-stabilization of a virtual topology for self-organization in ad hoc networks. 7th International Symposium on Self-Stabilizing Systems (SSS), Barcelona, Spain, October 2005.

[ThVa05b] F. Theoleyre and F. Valois. Routage hybride sur structure virtuelle dans les réseaux mobiles ad-hoc. Colloque Francophone sur l'Ingénierie des Protocoles (CFIP), Bordeaux, France, March 2005.

[ThVa05c] F. Theoleyre and F. Valois. Virtual structure routing in ad hoc networks. IEEE International Conference on Communications (ICC), Seoul, Korea, May 2005.