PACMAN: Passive Autoconfiguration for Mobile Ad hoc Networks

Kilian Weniger, Universität Karlsruhe (TH), Germany.

Abstract— **Mobile ad hoc networks (MANETs) enable the communication between mobile nodes via multi-hop wireless routes without depending on a communication infrastructure. In contrast to infrastructure-based networks, MANETs support autonomous and spontaneous networking and, thus, should be capable of self-organization and -configuration. This paper presents PACMAN, a novel approach for the efficient distributed address autoconfiguration of mobile ad hoc networks. Special features of PACMAN are the support for frequent network partitioning and merging, and very low protocol overhead. This is accomplished by using cross-layer information derived from ongoing routing protocol traffic. E.g., address conflicts are detected in a passive manner based on anomalies in routing protocol traffic. Furthermore, PACMAN assigns IP addresses in a way that enables their compression, which can significantly reduce the routing protocol overhead. The performance of PACMAN is analyzed in detail based on various simulation results.**

Index Terms— **Mobile ad hoc networks, address autoconfiguration, Passive Duplicate Address Detection (PDAD), PACMAN.**

I. Introduction

Most research efforts in the area of mobile ad hoc networks focus on efficient routing. The proposed protocols can be classified in proactive or table-driven, and reactive or on-demand approaches. While the former continously maintain routes to all nodes in the network, the latter only discover routes when needed. With either approach, nodes must be configured with a unique network layer address before a valid loop-free route can be established and unicast communication among nodes becomes possible. In order to support spontaneous networking, the address assignment should be automatically done. Address autoconfiguration protocols for traditional networks, however, cannot be applied to mobile ad hoc networks in general for several reasons (see section II). Thus, new protocols have to be developed.

Besides the dynamic multi-hop topology, MANET protocols have to deal with limited resources like bandwidth and energy, the shared medium, and an increased packet error rate as a result of the wireless channel's properties. Since control traffic, e.g., of the routing protocol is usually transmitted in-band, the protocol overhead often limits the amount of data traffic the network can handle. Also, the number of transmitted control packets limits the lifetime of the nodes' batteries. Hence, low protocol overhead is one of the most important design goals for MANET protocols.

We propose a novel approach for the distributed address autoconfiguration of mobile ad hoc networks, which we call Passive Autoconfiguration for Mobile Ad hoc Networks (PACMAN). PACMAN generates almost no protocol overhead, since it uses information from ongoing routing protocol traffic. This cross-layer/cross-protocol design is much more efficient in terms of protocol overhead than a design, which is independent of the routing protocol. A modular architecture eases the integration of new routing protocols.

A significant challenge to autoconfiguration protocols is the potentially frequent merging of independently configured networks. As a consequence, address conflicts may occur and influence the routing of data packets. PACMAN supports frequent network partitioning and merging, and can efficiently resolve address conflicts in a timely manner whenever they occur.

The rest of the paper is organized as follows: an overview of issues and current research efforts in the area of address autoconfiguration of MANETs is given in section II. The system architecture of PACMAN is presented in section III. The most important components of PACMAN are described in the following sections IV to IX. To analyze the performance of PACMAN, extensive simulation experiments were conducted. Some of the corresponding results are discussed in section X. Finally, section XI concludes the paper and gives directions for future research.

II. Issues and Related Work

Autoconfiguration protocols for traditional IP-based networks can be classified in stateless and stateful approaches. In case of stateful approaches, a central entity assigns addresses to new nodes. Because this entity keeps state information about addresses that have already been assigned, address conflicts do not occur. An example is the Dynamic Host Configuration Protocol (DHCP) [1].

In contrast, stateless approaches are decentralized: a new node chooses an address by itself and checks its uniqueness using a Duplicate Address Detection (DAD) procedure. The node may, for instance, send a query to the chosen address. If it does not receive a response, it assumes that the address has not yet been assigned to another node. In this case, the node configures its network interface with this address and defends it against other nodes which want to use this address.

As the nodes do not keep state information about which addresses are already assigned, they can only randomly choose an address or embed a globally unique hardware ID. Due to the small size of the IPv4 address space, the latter usually makes

only sense in case of IPv6. Examples of stateless approaches are the IETF Zeroconf protocol [2] and the IPv6 stateless address autoconfiguration (SAA) protocol [3].

One may argue, that providing unique addresses is very easy if the size of the address space is large enough to embed a globally unique hardware ID (as in case of IPv6). Unfortunately, there is no hardware ID which is truly globally unique. E.g., IPv6 SAA uses the 48 bit IEEE MAC address, but some manufacturers sell network adapters with non-registered MAC addresses or MAC addresses may get corrupted during the manufacturing process. Furthermore, most network adapters allow users to change the MAC address to arbitrary values. Some network adapters do not have a IEEE MAC address at all, e.g. in sensor networks. Another drawback of hardware ID-based addresses are privacy concerns. E.g., [4] proposes to use a random interface identifier instead of the IEEE MAC address. Hence, IPv6 SAA performs a DAD to ensure address uniqueness.

Autoconfiguration protocols for traditional IP-based networks cannot be applied to MANETs without changes. DHCP requires a central entity, which may not always be reachable in a MANET due to its highly dynamic topology. Thus, a distributed approach is required. Stateless protocols like Zeroconf or SAA are distributed, but are designed for LANs and require all nodes to be reachable via single-hop broadcast messages to query the chosen address. Since MANETs are multi-hop networks, not all nodes can be reached via such a broadcast message.

A major challenge is the handling of potentially frequent network mergers. Because communication between nodes of different network partitions is not possible, no autoconfiguration protocol can provide address uniqueness across network partitions. If two partitions merge, an address conflict may arise, which must be resolved as quickly as possible.

Recently, some autoconfiguration protocols for MANETs have been proposed (see [5] for a survey). The first approaches were adaptations of autoconfiguration protocols for traditional networks. E.g. Perkins et al. propose an adaptation of the stateless IETF Zeroconf protocol for MANETs [6]. A node randomly chooses an address and performs a DAD by flooding the network with an Address Request (AREQ) message, which contains the chosen address. A node having the same address defends it by replying with an Address Reply (AREP) message, which is sent over the reverse path established by the AREQ message. If there is no other node in the network with this address, a timer at the originator node expires and the address is considered unique. In the following, we call this mechanism Query-based DAD. A serious drawback of this approach is that network merging is not supported.

Another DAD mechanism called Weak DAD (WDAD) is proposed in [7]. Other than the Query-based DAD, WDAD is integrated with the routing protocol and can continuously detect duplicate addresses with information added to routing protocol packets. Thus, the routing protocol packet format has to be modified. The main idea is to add a key to each address that is distributed by the routing protocol. The key can be of arbitrary length and is chosen once by each node either randomly or based on a Universal Unique ID (UUID). A node detects a conflict if it receives two address-key-pairs with the same address, but different keys. Thus, a conflict cannot be detected if two nodes choose the same address and the same key. In case of random keys, the probability of an undetectable conflict decreases with increasing key length. Since the key length determines the additional overhead, there is a trade-off between routing protocol overhead and the probability that some conflicts cannot be detected.

An example of a stateful approach is MANETconf [8]. Using MANETconf, each configured node is able to assign addresses to new nodes and therefore maintains an allocation table of already assigned addresses in the network. A new node called “requester” searches for an already configured node called “initiator” by sending a special broadcast message. The initiator replies, chooses an unassigned address and ensures the uniqueness of this address by a mutual exclusion algorithm. It floods a special message and asks all nodes in the network for permission to assign this address. The address is only assigned if all nodes send a positive reply. If a node does not reply at all (after repeated requests), the initiator assumes that this node has left the network. Thus, its address is removed from the allocation table.

The challenging part is the handling of network mergers. The authors propose to identify each partition by a partition ID. This ID is composed of the smallest address in the network and a UUID that is provided by the node with the smallest address. The partition ID is included in periodically transmitted messages. A node detects a partition merger by receiving a message with a different partition ID. In this case, both nodes exchange their allocation tables and flood them in their partition. Each node that finds its address in the allocation table of the other partition has to give up its addresses.

A problem of this kind of indirect DAD is that it is based on global states (the allocation table). In case of inconsistencies, unnecessary address changes may occur or address conflicts may be undetectable. This may happen, e.g., when packets get lost during the exchange of the allocation tables after a network merger. Thus, the main challenge of the stateful approaches is a reliable global state synchronization. This requires reliable message exchange including a reliable broadcast mechanism and results in considerably complex protocols with high protocol overhead.

A similar approach is used in [9]. Here, the author proposes a new network layer addressing scheme with variable-length addresses. Such a scheme has only been proposed for MAC-layer addresses before [10]. An advantage of variable-length addresses is that the overhead of protocols sending a lot of addressing information can be reduced significantly, e.g., if only 4 bit are used to address a network of 15 nodes. But Internet applications and routing protocols use IP addresses and, thus, cannot be used with the new addressing architecture. The author proposes to use an Internet gateway to translate MANET addresses into IPv6 addresses for the communication with Internet nodes, but he does not propose a mechanism to enable IP-based communication within the MANET.

In [11], nodes maintain disjoint allocation tables to be able to autonomously assign addresses. However, global synchronization is still required to guarantee disjointness.

In summary, the main challenge of address autoconfiguration in MANETs is the provision of both address uniqueness in the presence of frequent network partitioning and merging and efficiency in terms of protocol overhead. Current approaches are nonsatisfying with respect to these requirements.

III. System Architecture of PACMAN

PACMAN is a novel approach for the efficient distributed address autoconfiguration of MANETs. It uses cross-layer information from ongoing routing protocol traffic and utilizes elements of both the stateful and the stateless approach. It can thus be classified as a hybrid approach. State information about addresses assigned in the network is collected in a lazy manner to save bandwidth. As in stateless protocols, a node assigns an address to itself[1].

We propose a modular architecture for PACMAN (see figure 1). A *routing protocol packet parser* extracts information from incoming routing protocol packets and hands them to the *PACMAN manager* in a generic format, which in turn delegates the information to the respective components. The protocol parser is modular itself to support different routing protocols. The *address assignment* component selects an address using a probabilistic algorithm (see section V). It also maintains the allocation table. Potential address conflicts, e.g. occurring after two networks merged, are detected by the *Passive Duplicate Address Detection (PDAD)* component (see section VI and VII). PDAD does not send any control packets, instead PDAD algorithms analyze incoming routing protocol packets to derive hints about conflicts. Again, a modular approach is used to support different routing protocols[2] and algorithms. In case a node detects a conflict of another node's address, the *conflict resolution* component notifies the respective node (see section VIII), which can then change its address to resolve the conflict. An *address change management* component can additionally notify communication partners about an address change to prevent transport layer connections from being broken (see section IX). To reduce the size of routing protocols packets, IP addresses in such packets can be compressed to variable-sized addresses by the *address encoding* component before they are given to the MAC layer (see section IV). Encoded addresses in incoming routing protocol packets are decoded back to fixed-sized IP addresses. Hence, the compression is transparent to the network layer and above. However, this component is an optional enhancement. PACMAN also works without address encoding.

In the following sections, the specific components are discussed in detail. Two optional components that are not discussed in this paper are the *clustering component* for providing an address hierarchy [12] and the *passive address resolution (PAR) component* for deriving IP-MAC-address mappings from incoming routing protocol packets as an optimization for ARP/NDP (see [13] for details).

[1]Here, we assume that each node only has one network interface. However, PACMAN can be extended to support multiple addresses per node.

[2]The first level separates proactive (PR) and reactive (RR) routing.

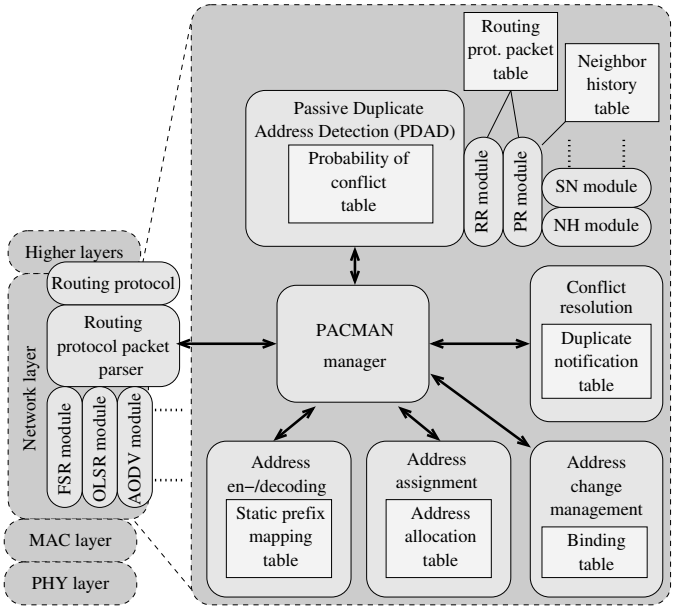


Fig. 1. Modular architecture of PACMAN

IV. IP Address Encoding

The maximum number of nodes that can be uniquely identified in a network is determined by the size of the address space from which the addresses are picked. This size can also affect the size of the address representation and, therefore, the size of the protocol packets that contain addressing information. Subsequently, an oversized address space means a waste of bandwidth. A variable-sized address space growing with the current network size would be advantageous.

As was already mentioned in section II, [9] proposes a new network layer addressing architecture with variable-sized addresses. A drawback of this addressing architecture is that it is not compatible with the IP addressing architecture. Our approach is to assign IP addresses in a manner that enables IP address compression. The basic idea is to encode addresses in outgoing routing protocol packets below the network layer to lower the protocol overhead on the wireless channel. Addresses in incoming routing protocol packets are decoded back to common IP addresses to remain compatibility with the IP addressing architecture.

For communication within the MANET, IP addresses with the so-called MANET-local prefix[3] can be used. Since MANET-local addresses are not globally routeable, they only have to be unique within the MANET, and may be reused in other network partitions. For this purpose, the IPv4 and IPv6 address space is clearly oversized: 65536 or $\approx 1.8 \cdot 10^{19}$ nodes can be addressed, respectively. Once assigned, the MANET-local addresses can also be used as a basis to build unique, globally routeable addresses for communication with nodes in the Internet, e.g. by replacing the prefix.

We introduce a variable-sized address space, which we call the *virtual address space*. The address assignment component chooses an address from this virtual address space. This

[3]In [6], 169.254/16 for IPv4 and fec0:0:0:ffff::/64 for IPv6 are used

address has a *virtual address length* that can be at most as long as the fixed length of the IP address minus the prefix (16 bit in case of IPv4 and 64 bit in case of IPv6 when using the MANET-local prefix). E.g., in a network of 60 nodes, a virtual address space size of 6 bit would be enough to uniquely address all nodes.

A possible address encoding scheme is the well-known Huffman coding, as used in [14] for MAC addresses. But since a dynamic mapping between IP addresses and codewords has global meaning, and some kind of global synchronization would be required. As already mentioned, such a synchronization is potentially very complex and may require a considerable amount of bandwidth.

Our approach is simpler and does not require global parameters: only part of an IP address, the virtual address space, is used as the node identifier; the remaining bits are unused and set to zero. Such an address can be easily compressed, e.g., by the well-known run-length coding (RLC). The size of the encoded address depends on the virtual address space, and the encoded address is self-contained: it can be decoded without any additional information.

The size of the encoded address can be further reduced by replacing frequently occurring prefixes by short codewords. As already mentioned, a dynamic mapping between addresses or prefixes and codewords is not desired. Thus, we propose to use a static mapping table and replace only well-known prefixes like the MANET-local prefix.

To clarify the encoding procedure, figures 2(a) and 2(b) show an example, where the 32 bit IPv4 address 169.254.0.13 is encoded to an 8-bit representation using RLC and prefix replacement. The size of the virtual address space is 4 bit in this example. In case of IPv4, the size of the encoded address is 8 bit, which is a reduction of 24 bit. An IPv6 address can be reduced to about the same size, which corresponds to a reduction of 120 bit in length.

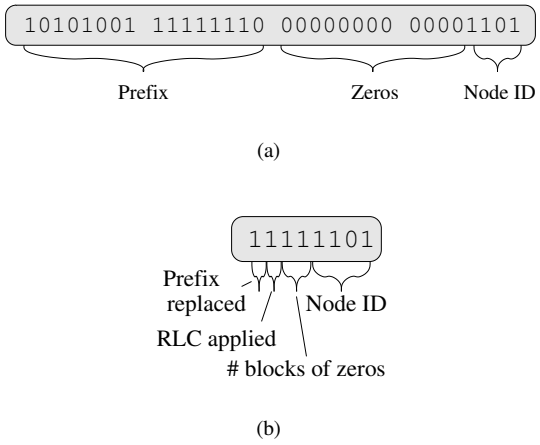


Fig. 2. Binary representation of decoded (a) and encoded (b) IPv4 address

Note that the RLC is not done bit-wise, but block-wise in this example. Therefore, the address space is divided in blocks (here: 4 bit blocks), which can increase the compression rate, because the number of blocks of zeros is smaller than the number of zeros. On the other hand, some unencoded bits may not be used as node identifier if the virtual address space size is not an integer multiple of the block size.

To be able to decode the routing protocol packets properly, all nodes have to understand the encoding scheme in use. To support different types, an additional encoding header can be used. If some nodes do not support address encoding at all, further mechanisms are required, e.g. the transmission of encoded packets on another logical transport channel, e.g., another UDP port. Moreover, the IP address encoding could as well be applied to addresses in other protocols, including IP itself.

V. Address Assignment

A node running PACMAN assigns an address to itself using a probabilistic algorithm. Based on a pre-defined conflict probability, an estimation of the number of nodes and an allocation table, the algorithm calculates the size of the virtual address space, randomly selects an address from this space and ensures that the address has not already been assigned according to the local allocation table. The selected address is assigned immediately. Hence, the node configuration time is always fixed (in the order of milliseconds), but the address may be duplicate with a certain probability. Due to the allocation table, this probability usually is zero. Only if many nodes join the network simultaneously, the allocation table is not up-to-date and conflicts may occur. However, they are detected and resolved by the PDAD component in a timely manner.

In the following, we show how the algorithm calculates the virtual address space size. First, we evaluate the likelihood of an address conflict in case of random assignments. The conflict probability only depends on the size of the address space and the number of nodes in the network: the larger the address space, the lower the conflict probability. This probability can be calculated in analogy to the well-known birthday paradox and is expressed by equation 1 with n being the number of nodes and r size of the address space (see [13] for a derivation).

$$P(E_c) = 1 - e^{-n}\left(1 - \frac{n}{r}\right)^{n-r-\frac{1}{2}} \quad (1)$$

Figure 3 shows a plot of this equation. In case of an address space size of 16 bit, which is e.g. used in case of IPv4 MANET-local addresses, the conflict probability is as high as $\approx 50\%$ in a network of 300 nodes.

With respect to address encoding, the choice of the address space size is a trade-off between compression rate and conflict probability. The larger the address space, the lower the conflict probability and the lower the compression rate.

Because the virtual address space size does not depend on a global state, and because each node assigns an address only to itself, each node can individually choose its virtual address space size. Regarding the desired conflict probability as a pre-defined Quality-of-Service parameter of the network and given that the number of nodes in the network is known, the optimal virtual address space size can be calculated at each node using equation 1. E.g. in a sensor network, nodes may tolerate a higher conflict probability if energy can be saved

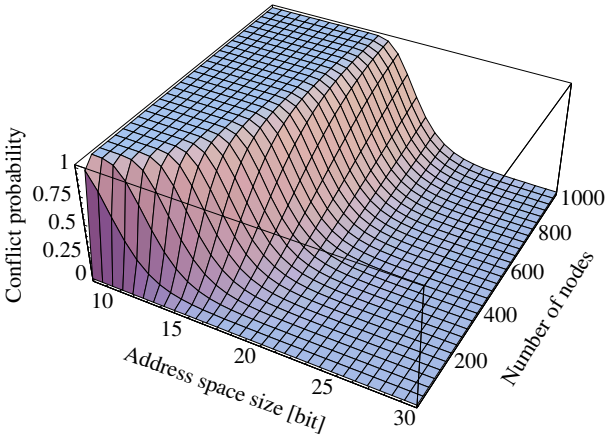


Fig. 3. Conflict probability in case of random address assignment

due to a higher compression rate. It may also be reasonable to allow a small fraction of nodes to use a larger virtual address space. E.g., nodes providing important services in the network may want to achieve a lower conflict probability and in return accept a lower compression rate for their addresses.

The conflict probability is further reduced by incorporating the address allocation table, which again is maintained using cross-layer information from ongoing routing protocol traffic. To speed up the configuration time, a new node can request the address allocation table from already configured neighbors. However, there can be a considerable amount of unknown or *hidden allocated addresses* in the network, e.g. if nodes join the network, but the routing information about them has not yet been distributed throughout the network. This especially is an issue for routing protocols that do not distribute routing information by flooding. A lot of hidden allocated addresses also occur immediately after many nodes simultaneously join the network, e.g. after a network merger.

Considering the allocation table, equation 1 does not correctly express the conflict probability anymore. The number of known allocated addresses in the address allocation table just reduces the size of the address space, from which the addresses are randomly chosen. Equation 2 gives an estimation of the conflict probability when using an allocation table with j being the number of hidden allocated addresses, r the virtual address space size and n the number of nodes in the network (for a derivation see [13]).

$$P(E_c) \quad \approx \quad 1 - (r \cdot e)^{-j} \frac{(r-n+j)^{r-n+j+\frac{1}{2}}}{(r-n)^{r-n+\frac{1}{2}}} \qquad (2)$$

j can be estimated, e.g., based on the allocation table. For $j = n$ the allocation table is empty. In this case, equation 2 equals equation 1. For $j = 0$, all allocated addresses are known and the probability of a conflict is zero.

VI. Passive Duplicate Address Detection

Duplicate addresses can occur, e.g., after two independently configured ad hoc networks have merged. To ensure proper unicast routing, these conflicts must be detected and resolved in a timely manner. PACMAN uses Passive Duplicate Address Detection (PDAD) [15] to detect duplicate addresses. Using PDAD, a node analyzes incoming routing protocol packets to derive hints about address conflicts. Thus, PDAD does not consume any additional bandwidth. The basic idea is to apply various PDAD algorithms to incoming routing protocol packets, which exploit protocol events that either

1) never occur in case of a unique address, but always in case of a duplicate address or
2) seldomly occur in case of a unique address, but often in case of a duplicate address.

In the first case, a conflict is certain if the event occurs once, whereas in the second case, a conflict is present only to a certain probability. In this case, long-term monitoring may be necessary. Such PDAD algorithms are probabilistic.

Each node maintains a table with the conflict probability for each address (see figure 1). Each time a PDAD algorithm identifies a hint about a conflict, it modifies the probability for the corresponding address in the table. If a certain threshold is reached, PDAD assumes that the address is indeed duplicate and triggers the conflict resolution.

The PDAD algorithms derive information about a sender's routing protocol state at the time the packet was sent from incoming routing protocol packets. This state can be compared to the state of the receiver or to another node's state obtained from a previously received packet from this address. Hence, each node stores information obtained from the last routing protocol packet received from a specific address in a routing protocol packet information table (see figure 1).

The receiver must have information about the time a specific routing protocol packet was sent to reason about conflicts. Without synchronized clocks and additional information in the packets, this time can only be estimated. We assume that the time interval during which a specific routing protocol packet is distributed in the network is bounded by the time span t_d, and that t_d can be estimated. In this case, routing information received by a node can never be older than t_d. Usually this assumption holds, because most ad hoc routing protocols use a duplicate cache and maintain routing information in a soft-state manner, and because the queuing and media access delays are bounded. Furthermore, most ad hoc routing protocols implicitly assume a certain maximum distribution time, otherwise they would not be able to distinguish fresh from stale routing information after sequence number wrap-arounds.

A. PDAD algorithms for proactive link-state routing protocols

In [15], we propose and discuss three algorithms for detecting duplicate addresses with proactive link-state routing protocols. The algorithms were first developed for a classic (proactive) link-state routing protocol model and were then applied to existing link-state routing protocols for MANETs with all their optimizations. In the following, these algorithms together with a couple of new algorithms are presented. As will be shown later, some of them can also be applied to other types of routing protocols.

In the classic link-state routing model we used, each node periodically issues link-state packets. These packets contain

the originator address, a sequence number, and a set of link-states consisting of the addresses of all neighbors. The link-state packets are flooded in the network. Each node forwards the packets on the application layer. A duplicate cache is used to prevent redundant forwarding.

1) PDAD-SN: This algorithm exploits the Sequence Numbers (SN) in the routing protocol packets to detect duplicate addresses in the network. The assumptions are that each node uses a sequence number only once within the time interval t_d, and that each node increments its own and only its own internal sequence number counter.

The algorithm is best explained by means of an example: figure 4 shows a scenario where node A and E have the same address 1. If node E receives a routing protocol packet with an originator address (OA) that is equal to its own address (1) and a slightly lower sequence number than its own sequence number (9), it cannot decide if this packet has recently been sent by itself or by another node with the same address.

In contrast, if node A receives a packet with originator address 1 and a sequence number (9) that is higher than its internal sequence number (1), this packet can only have been sent by another node with the same address[4]. Therefore, node A can detect a conflict of its address.

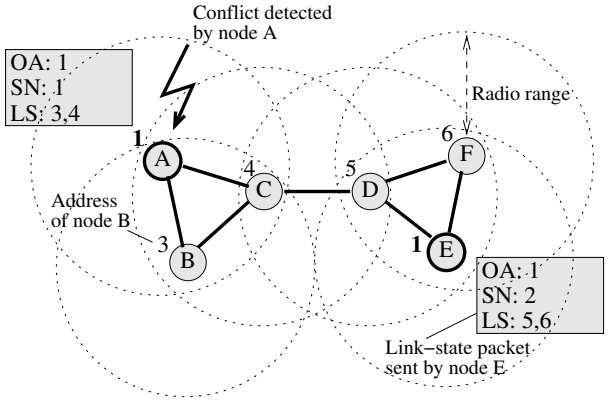


Fig. 4. Example of conflict detection using PDAD-SN

2) PDAD-SND: A conflict can as well be detected based on the Sequence Number Difference (SND). In the scenario illustrated in figure 5, node A and E again share the same address, but this time, both nodes have considerably differing sequence numbers: the difference is higher than the maximum possible incrementation within $t_1 - t_2 + t_d$, with t_1 and t_2 being the points in time when packet 1 and 2 were received, respectively. As a consequence, these packets could not have been sent by the same node. Intermediate nodes hence can conclude that the originator's address is duplicate.

3) PDAD-SNE: In the scenario shown in figure 6, node A and E have the same sequence number. If an intermediate node receives packets from both nodes, the originator address as well as the Sequence Numbers Equal (SNE). Therefore, the packets should be the same, forwarded by different nodes. But since the link-states (LS) in the packets are different, the packets obviously are not the same, and intermediate nodes thus can conclude that the originator address is duplicate.

[4]An exception is a sequence number wrap-around. See [15] for details.

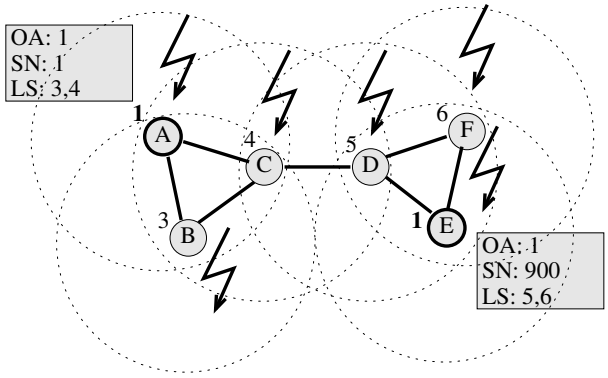


Fig. 5. Example of conflict detection using PDAD-SND

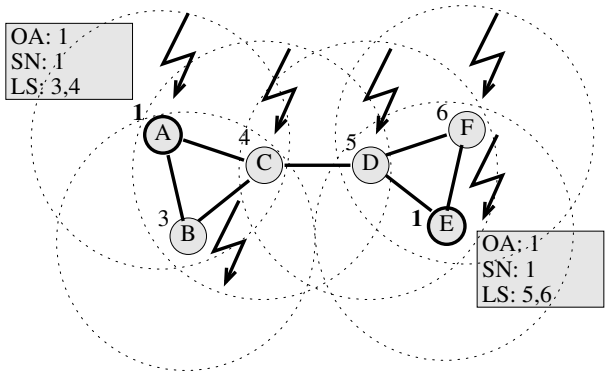


Fig. 6. Example of conflict detection using PDAD-SNE

4) PDAD-NH: The basic idea of the Neighborhood History (NH) algorithm is to exploit bidirectional link-states. If a node receives a link-state packet with its own address part of the set of bidirectional link-state of the originator, the originator must have been a neighbor of this node during the last period of time t_d. Otherwise, another node has the same address. To be able to make a decision, all nodes have to record their recent neighborhood history in an NH table. Figure 7 shows an example, where both node A and E have address 1. Node A receives a link-state packet from node F, which is a neighbor of node E. Therefore, node F's link-state packet contains address 1. Because node A has recorded the addresses of all recent neighbors, it can determine that the originator address (6) has not been a neighbor. Thus, it concludes that its address is duplicate.

5) PDAD-LS: The Link-State (LS) algorithms exploits that a node receiving a link-state packet with its own address as originator address must have had all link-state addresses as neighbors during the last period of time t_d. This condition can be verified using the NH table.

6) PDAD-LP: The assumption of the Locality Principle (LP) algorithm is that the link-states of a specific node usually do not completely change within one update interval of the routing protocol. This assumption holds up to a certain ratio of the node's relative speed to the neighboring nodes and the update interval. In contrast to the algorithms described so far, PDAD-LP is a probabilistic algorithms. Hence, PDAD-LP can only give hints about the probability of a conflict. Of course, at

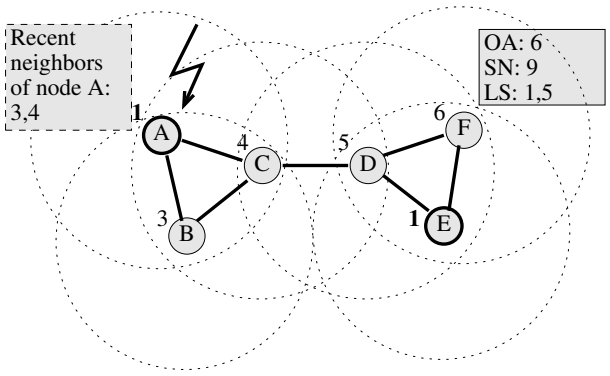


Fig. 7. Example of conflict detection using PDAD-NH

some point of time a decision about the presence of a conflict has to be made.

When a node receives a packet, it compares the set of link-states with that of the last received packet bearing the same originator address. To aquire a normalized conflict probability, equation 3 can be used with ls_1 and ls_2 being the set of link-states in two packets with the same originator address. The lower the number of link-states that match, the higher the conflict probability.

$$p_i = 1 - \frac{|ls_1 \bigcap ls_2|}{|ls_1| + |ls_2|} \tag{3}$$

Due to possible packet reordering, this probability can temporarily be high even if no conflict is present. To alleviate this problem, p_i can be smoothed, e.g. using an EWMA filter. Conflict resolution is triggered if p_i exceeds a threshold. Furthermore, only packets arriving with a short inter-arrival time slightly higher than the update interval should be considered.

7) PDAD-SNI: Routing protocol packet reordering usually does only happen due to forwarding. Hello packets sent by a specific node are received in the order they are sent. Since the Sequence Numbers always Increment[5] (SNI), a node can detect a conflict in the 2-hop neighborhood if it receives two hello packets from the same address with the second packet containing a lower sequence number than the first.

8) PDAD-SA: The Source Address (SA) algorithm utilizes information in the IP header of the routing protocol packets. If the routing protocol forwards packets on the application layer, the IP source address always is the address of the last forwarder. Thus, a conflict with a neighboring node can be detected if the IP source address of a received routing protocol packet is equal to the receiver's address.

9) PDAD-DC: The Duplicate Cache (DC) algorithm utilizes the IP header of the routing protocol packets as well. Since each routing protocol packet with a specific tuple (OA, SN) is only forwarded once and the forwarding takes place on the application layer, a conflict between two neighboring nodes can be detected if two packets with the same sequence number, originator address and IP source address are received.

[5]The only exception is a wrap-around

B. PDAD algorithms for on-demand routing protocols

Passive Duplicate Address Detection can also be applied to on-demand routing protocols. Due to their passive nature, the DAD is also on-demand, and can only detect conflicts between nodes involved in a route discovery or maintenance procedure.

In the following, we define a model of an on-demand routing protocol and present some PDAD algorithms for this model. The model can be regarded as a simplified version of AODV [16]: a node floods the network with a Route Request (RREQ) message to discover a route to a specific destination. This message is forwarded by intermediate nodes on the application layer. Unlike nodes running AODV, only the destination node may answer with a Route Reply (RREP) message, which is routed over the reverse path of the RREQ. The RREQ contains an RREQ ID to identify a specific message and to enable the employment of a duplicate message cache. A single RREQ is only processed once by each node. Both messages contain sequence numbers to assure loop freedom. The route is maintained by periodically sending RREQ messages as long as it is active.

PDAD-LP, -LS and PDAD-NH, of course, cannot be applied, because there is no link-state information in the packets. PDAD-SA and -DC as well as -SN, -SNE and -SND can be applied.

In the following, special algorithms for on-demand protocols are presented.

1) PDAD-RNS: The RREQ-Never-Sent (RNS) algorithm detects a conflict if an RREQ is received with the receiver's originator address, but an RREQ has never been sent for this destination. In this case, another node with the same address must have sent the RREQ.

2) PDAD-RwR: The RREP-without-RREQ (RwR) algorithm detects a conflict if a node receives an RREP for its own address, but an RREQ has never been sent for this destination.

3) PDAD-2RoR: Due to the application of a duplicate message cache, a singe destination node only replies once to a specific RREQ. The 2RREPs-on-RREQ (2RoR) algorithm detects a conflict if more than one RREP has been received from the destination address of a specific RREQ.

VII. Applicability to AD HOC routing protocols

The PDAD algorithms presented is the last section only apply to the above defined models. In this section, we discuss their applicability to the routing protocols FSR, OLSR and AODV.

A. PDAD and FSR

The Fisheye State Routing (FSR) [17] protocol is a proactive link-state routing protocol with two special properties: the aggregation of link-state packets and the application of the fisheye technique. Unlike most link-state routing protocols, FSR does not flood link-state packets. Instead, only one aggregated routing protocol packet per update interval is sent by each node, similar to the technique used by distance-vector routing protocols. In order to prevent unlimited distribution, topology information is held in a soft-state manner and is only distributed if it is fresh or if a neighbor has stale information.

The link-state packets also serve the purpose of neighbor sensing. The fisheye technique was introduced to reduce the protocol overhead: routing information of far-away nodes are sent less frequently than information of nearby nodes.

The maximum distribution time t_d of routing information is much higher than in case of routing protocols that distribute routing information by flooding. Hence, PDAD may require more time to detect a conflict. PDAD-LP may trigger false alarms due to the slow update frequency for far-away nodes. Because the link-states in FSR packets are unidirectional, PDAD-NH cannot be applied. In contrast, PDAD-SN, -SNE and -SND can be applied and are able to detect all conflicts. Three cases can be distinguished:

- Two nodes sharing the same address have considerably different sequence numbers. In this case, the conflict can be detected by an intermediate node using PDAD-SND or by the node with the lower sequence number using PDAD-SN.
- The sequence numbers of two nodes having the same address are almost equal. The conflict can then be detected by the node with the lower sequence number using PDAD-SN.
- Two nodes with the same address have the same sequence numbers. The conflict can be detected by an intermediate node using PDAD-SNE.

PDAD-SA can accelerate the detection in some cases. In summary, we recommend a combination of PDAD-SA, -SN, -SNE and -SND for FSR.

B. PDAD and OLSR

The Optimized Link-State Routing protocol (OLSR) [18] distributes so-called Topology Control (TC) packets containing unidirectional and bidirectional link-states in the network. Other than FSR, OLSR uses hello messages for neighbor sensing and floods TC messages. A special feature of OLSR is the employment of Multi-point Relays (MPR): every node selects certain neighbors as MPR nodes. A node selects at least those 1-hop neighbors that allow the node to reach all 2-hop neighbors. MPR nodes only forward TC messages sent by their selectors. A node that has not been selected as MPR by anyone does not issue TC messages at all. Other nodes issue TC messages, but only declare their MPR selectors. Using the MPR technique, a considerable amount of protocol overhead can be saved.

In case of OLSR, PDAD-LP may trigger false alarms, because the link-states only are a subset of the neighborhood, which can change completely due to a change in the MPR selection. Since TC messages contain bidirectional link-states, PDAD-NH can be applied. PDAD-LS, -SN, -SNE and -SND can be applied as well. We recommend to combine these algorithms for the following reason: as already mentioned, non-MPR nodes do not issue link-state packets, but PDAD-SN, -SNE and -SND only detect conflicts of nodes that *do* issue link-state packets. PDAD-NH on the other hand requires a neighbor of a node to issue a link-state packet. Since OLSR uses application layer forwarding, a duplicate cache and hello messages, PDAD-SA, PDAD-DC and PDAD-SNI can be applied as well. To accelerate the conflict detection, all algorithms can also be applied to hello messages.

A challenge is that duplicate addresses can affect the MPR selection. If too few nodes are selected as MPRs, routing protocol messages may not be propagated throughout the entire network and data packet routing may fail. However, this is only a problem in static scenarios with conflict partners less than 5 hops away from each other. An example is shown in figure 8. Node A and D have the same address. As a consequence, both node B and C do not have any real 2-hop neighbor, i.e. one that is not 1-hop neighbor at the same time. Thus, they select none of their neighbors as MPR. Node A and D do not receive routing information from each other and the conflict cannot be detected by the algorithms presented so far.

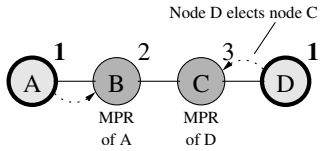


Fig. 8. Example of an address conflict influencing the MPR selection

A solution to this problem is that node B and C execute PDAD-NH on behalf of node A and D, respectively: if node B receives a TC message from node C containing the address of a neighbor of node C (here: node D with address 1), node B checks whether the originator address (here: 3) is contained in the hello packets received from address 1 (here: from node A). If this is not the case, it may conclude that there is a conflict of address 1. However, because node B does not know the complete neighborhood history of node A, it cannot be sure. It sends the information to node A, which can then make the final decision based on its NH table using PDAD-NH. The algorithm for the preliminary decision is referred to as the *extended NH (eNH)* algorithm.

However, there are scenarios with multiple conflicts, in which PDAD-eNH fails. Figure 9 shows such a scenario. Node B, C and D all have no real 2-hop neighbors and, thus, select none of their neighbors as MPR. The conflicts cannot be detected by PDAD-eNH, because the neighbors' addresses of the conflicting nodes are the same from node C's point of view. However, the conflict can be detected by extending PDAD-eNH and -NH: all nodes additionally store the link codes of transmitted hello messages in their NH tables. The TC messages that node C receives from node B have originator address 2 and contains address 1 as a link-state. Thus, node C can conclude that a node with address 1 must have selected a node with address 2 as MPR. Since the hello messages received from node D indicate that a node with address 1 has not selected a node with address 2 as MPR, node C can assume a conflict. Because the TC message can be delayed and node C is not aware of the complete neighborhood history of node D, it sends the information to node D, which can then make the final decision based on its NH table. In analogy to PDAD-eNH and -NH, these algorithms are referred to as *PDAD-eMPR* and *-MPR*.

Note that care must be taken regarding the bidirectional

link-states and the MPR selection after an address change, because both are based on the context of the old address. Hence, a node has to set all its bidirectional or symmetric link-states to asymmetric and remove all addresses from its MPR selector set. Without these modifications, PDAD-NH and -MPR would detect a conflict of the new address even if it is unique.

In summary, we recommend a combination of PDAD-SA, -DC, -SNI, -SN, -SNE, -SND, -LS, -eMPR and -MPR.

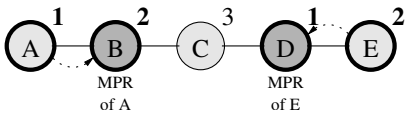


Fig. 9. Example of multiple address conflicts influencing the MPR selection

C. PDAD and AODV

The Ad hoc On-demand Distance-Vector (AODV) routing protocol [16] differs from the model defined in section VI-B in some aspects. The major differences are:

1) Intermediate nodes are allowed to reply to an RREQ if they have a fresh-enough route to the destination. Thus, PDAD-2RoR may only be applied if RREPs sent by intermediate nodes can be distinguished from RREPs sent by the destination. Since this is not the case in AODV, a new flag must be introduced to the RREP messages to allow the application of PDAD-2RoR.
2) The route is reestablished only if a link on an active route breaks. In this case, an intermediate node either performs a local repair procedure or sends a Route Error (RERR) message back to the source, which then can issues a new RREQ. Thus, if a route is active for a very long time and does not break, the destination sequence numbers for a specific destination may considerably differ from the sequence number at the destination itself. This means that the distribution time of routing information can be arbitrarily high, which is not only a problem for PDAD: a sequence number difference higher than half of the sequence number space may result in permanent routing loops in AODV even in case of unique addresses, because fresh routing information cannot be distinguished from stale routing information anymore. Hence, PDAD-SN, -SND and -SNE should only be applied to the RREQ ID or AODV must be modified to perform a synchronization of sequence numbers on active routes from time to time.
3) Because RREPs are also used as hello messages, the PDAD-2RoR algorithm may not be applied to RREPs with a broadcast address as destination address.

A major problem is the expanding-ring-search technique of AODV. Using this technique, a node first searches for a route to the destination address in the immediate vicinity and incrementally expands this search area. It aborts the search procedure if a node with the desired route information has been found. If the destination address is duplicate and a node having a route to this address is in the initial ring-search area, while a second node with routing information about the other node with the same address is not, the conflict cannot be detected and it cannot be determined whether the route found leads to the intended destination node. Thus, PDAD is only able to detect all conflicts if the expanding-ring-search technique is not used for the initial route discovery.

In this case, a combination of PDAD-RNS, -2RoR, -SN, -SND and -SNE is able to detect a conflict at the time an RREQ containing the duplicate address is sent. If node A and B both have the same address, there are three possible cases:

1) Neither node A nor node B issues an RREQ. Instead, a third node C issues an RREQ with the address of node A and B as destination address. In this case, node C can detect the conflict using PDAD-2RoR.
2) Either node A or node B issues an RREQ. In this case, one of them can detect the conflict using PDAD-RNS or -SN.
3) Both node A and B simultaneously issue an RREQ. In this case, they can detect the conflict using PDAD-SND, PDAD-SNE or -SN.

We recommend a combination of PDAD-SA, -DC, -SNI, -SN, -SNE, -SND, -RwR, -RNS and -2RoR for AODV.

VIII. Conflict resolution

As already discussed in section VI, a node can either detect a conflict of its own address or of another node's address. In any case, at least one node involved in the conflict must be notified, so that it can change its address to resolve the conflict. PACMAN issues so-called Address Conflict Notification (ACN) messages for this purpose, which are unicast to the duplicate address.

There are numerous possible strategies, which node should change its address, e.g., using the number of open transport layer connections or the period of time a node is part of the network. To prevent the distribution of additional information in the network, PACMAN uses a rather simple approach: an ACN message is sent in the direction from which the corresponding routing protocol packet was received (see figure 10). This way, the node that has joined the network most recently tends to change its address and no additional information must be distributed in the network. Moreover, a node detecting a conflict discards the corresponding routing protocol packet to prevent the distribution of inconsistent routing information.

It may happen that many intermediate nodes detect a conflict at almost the same time. In this case, many nodes unicast notifications to one node: the node with the duplicate address. This many-to-one communication is known as concast and is a problem especially in large ad hoc networks, because it can lead to increasing media access delays in the area of the destination node. Figure 11a illustrates the problem. Each unicast packet sent is represented by an arrow.

To alleviate this problem, a duplicate message cache is employed: if a node has recently forwarded a notification to a specific address, it does not forward notifications to this address anymore for a certain period of time. Thus, the number of medium accesses can be significantly reduced (see figure 11b).

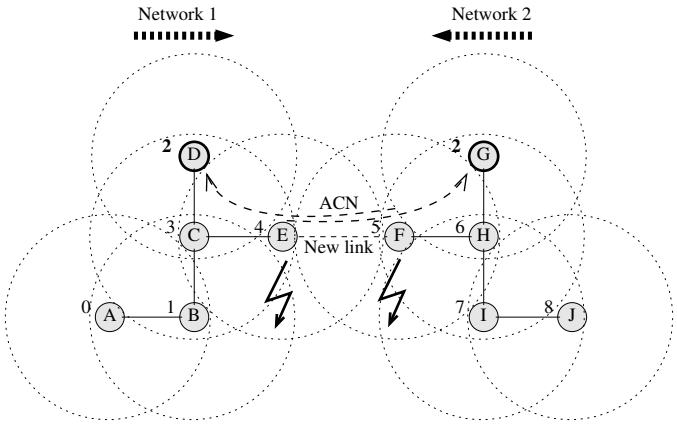


Fig. 10. Example of a conflict resolution using ACN messages

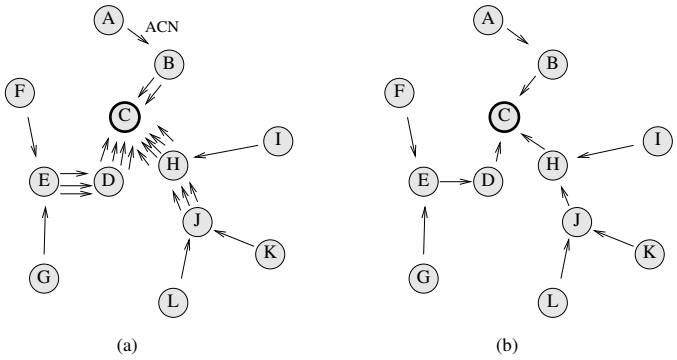


Fig. 11. Example of a concast communication without duplicate message cache (a) and with duplicate message cache (b)

IX. Address Change Management

Usually, transport layer connections are defined by the addresses of the endpoints and, thus, break if an endpoint changes its address. This problem is similar to the situation of network layer mobility, where a node moves to another subnet and has to change its address to a topologically correct one. For these scenarios, solutions like Mobile IP exists. But other than in infrastructure-based networks, permanently reachable home or foreign agents cannot be deployed in mobile ad hoc networks.

We borrow the idea of route optimization from Mobile IPv6 [19]: the node that changes its address notifies its communication partners about the new address using a Binding Update (BU) message. It declares its old address as home address. All nodes manage a binding table (see figure 1) and can use the routing header in case of IPv6 or an IP tunnel to send data packets destined for the old address to the new address. These packets are then de-capsulated at the destination and arrive at the transport layer with the old destination address. In case a location management or name service exists, it should also be notified about the new address.

X. Evaluation

PACMAN was implemented and evaluated in testbed[6] and simulation environment. In this section, the results of some simulation experiments done with an extended version of GloMoSim 2.03 [20] using FSR and OLSR as routing protocols are presented. The extensions include support for duplicate addresses and address changes during the simulation, logging of time dependent statistics etc. FSR are part of the GloMoSim distribution and were updated to current versions. OLSR is not part of the GloMoSim distribution and was implemented from scratch.

If not specified otherwise, the parameters of table I were used. Because we also wanted to analyze the effects of network partitioning and merging, a moderate to low node density was chosen (≈ 7 neighbors on average). Each simulation was repeated 20 times using different random-number generator initializations. The graphs show the mean average value, usually including a 90% confidence interval.

Parameter	**Value**
Routing protocols	FSR, OLSR
FSR update interval In/Out-Scope	$5s/15s$
OLSR update/hello interval	$5s/2s$
Number of nodes	50
Simulation area	$1500m \times 1500m$
Simulation time	$300s$
Mobility model	Random Waypoint
Min/Max speed/Pause time	$2\frac{m}{s}/4\frac{m}{s}/0s$
MAC protocol	802.11
Transmission range	$276m$

TABLE I
Default simulation parameters

A. Simulation Results for IP address encoding

We first analyzed the IP encoder regarding its efficiency by measuring the IPv4 and IPv6 routing protocol packet compression rate with respect to various parameters. All nodes had the same virtual address space size v, which was varied between 6 and 16 bit. The block size of the encoder was set to 2 bit. The simulation time was 600s.

Figure 12 shows the expected behavior: the packet compression rate is higher with a smaller virtual address space. Furthermore, the compression rate is substantially higher in case of IPv6, because there are much more unused bits than in case of IPv4. Regarding the routing protocols, the compression rate of FSR packets is higher than that of OLSR packets, because FSR packets contain a higher fraction of addressing information. The highest packet compression rate is 90%, which means that the compressed packet is 10% of the original size or 10 times smaller than the original packet.

Figure 13 shows, how the block size affects the compression rate (here for IPv4 FSR packets). With increasing block size, the number of bits needed to represent the leading zeros decreases, but more unused bits are left unencoded if the block size is not an integer multiple of the virtual address space size. Thus, the compression rate is limited to certain levels, whose number decreases with increasing block size. According to our results, a good compromise seems to be a block size of 2 bit.

[6]The source code is available at http://pacman-autoconf.sourceforge.net

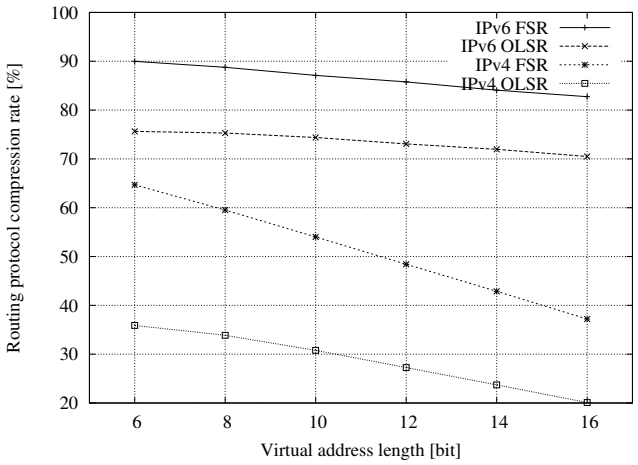


Fig. 12. Routing protocol packet compression rate vs. virtual address length

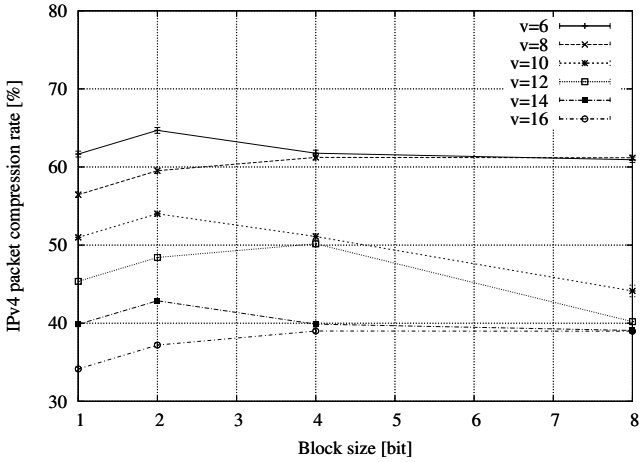


Fig. 13. IPv4 packet compression rate (FSR)

B. Simulation Results for PDAD

To analyze the performance of PDAD with FSR and OLSR, three scenarios with duplicate addresses were simulated. Initially, all nodes randomly chose an address. One or two conflicts (C) with two or four duplicate addresses (DA) appeared at 50s simulation time by forcing an address change to a previously unused address. All scenarios were simulated with the detection at intermediate nodes switched on (I) and off. In case of FSR, a combination of PDAD-SA, -SN, -SND and -SNE, in case of OLSR, a combination of PDAD-SA, -SN, -SNE, -SND and -NH was applied. The time span t_d was assumed to be lower than 200s. Because the sequence numbers affect the dissemination of routing information as well as PDAD-SN, -SND and -SNE itself, three initialization scenarios were considered: randomly chosen from the complete space (scenario 1), randomly chosen between zero and 100, which is assumed to be the maximum possible incrementation within t_d (scenario 2) and 0 (scenario 3).

The most important simulation result is that no false alarms occur and all conflicts are detected if intermediate conflict detection is switched on. The conflict detection time depends on the routing protocol packet distribution time and is in the order of seconds. Figure 14 shows the conflict detection time for the different OLSR scenarios with a simulation area of $1000m \times 1000m$.

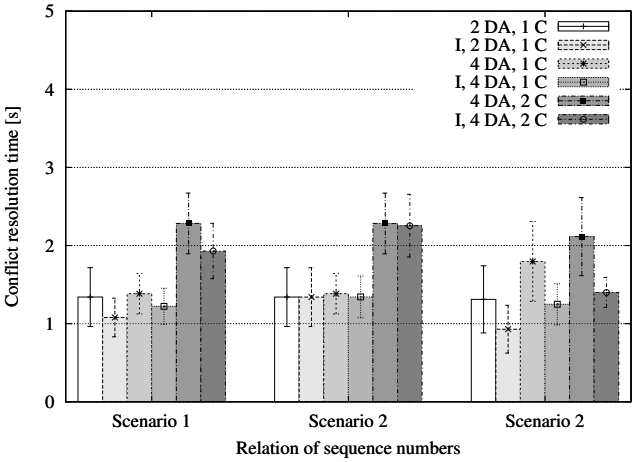


Fig. 14. Conflict resolution time using PDAD (OLSR)

C. Comparison of PDAD to other DAD approaches

Next, we compared PDAD to different DAD approaches, namely Query-based DAD and Weak DAD, by replacing the PDAD module of PACMAN. The number of nodes was varied from 25 to 200 nodes, all moving according to the random waypoint model. To keep the network density at a reasonable level, a simulation area of $1500m \times 1500m$ for 25 to 100 nodes and $2500m \times 2500m$ for 125 to 200 nodes was used, respectively. In all scenarios, 10 nodes have a duplicate address at the beginning of the simulation. The routing protocol was FSR and the simulation time 300s. The key length used for WDAD was 32 bit. The Query-based DAD repeats the query three times with a random interval between 0 and $500ms$.

As figure 15 shows, the Query-based DAD can only detect all 10 duplicate addresses in scenarios with a high node density, because it performs the DAD only once at the beginning of the simulation. In this case, the conflicts are detected very quickly (see figure 16). Weak DAD and Passive DAD detect all conflicts in all scenarios. The resolution times for both approaches are almost the same and depend on the time the routing protocol needs to distribute the routing information. In large networks or if the network density is low, this takes more time than in small, densely populated networks.

Figure 17 shows the protocol overhead generated within the simulation time. Note that the graph also includes the routing protocol overhead, which increases with the average number of neighbors and the network size. WDAD performs worst. The 32-bit keys distributed with every address in every routing protocol packet almost doubles the routing protocol overhead. The Query-based DAD has the second-lowest overhead, although it triggers the DAD only once at the beginning of the simulation. In contrast, PDAD does not generate any protocol overhead. Only the notification of nodes having a duplicate address causes the transmission of control packets.

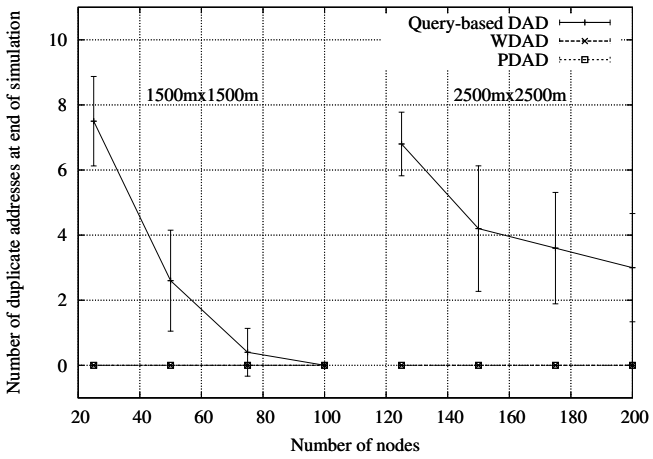


Fig. 15. Number of duplicate addresses failed to detect by different DAD approaches (FSR)

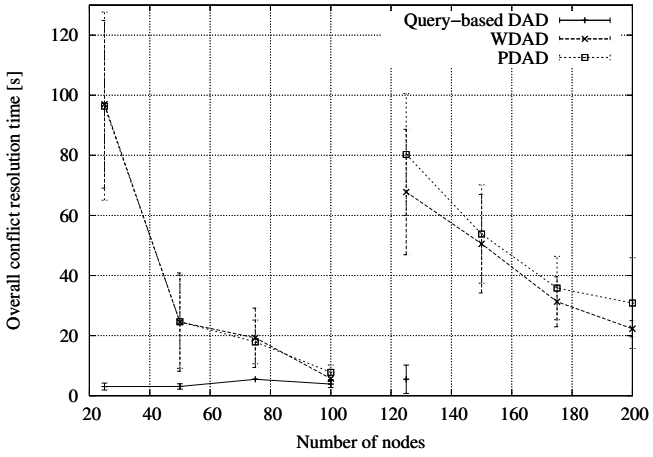


Fig. 16. Overall conflict resolution time with different DAD approaches (FSR)

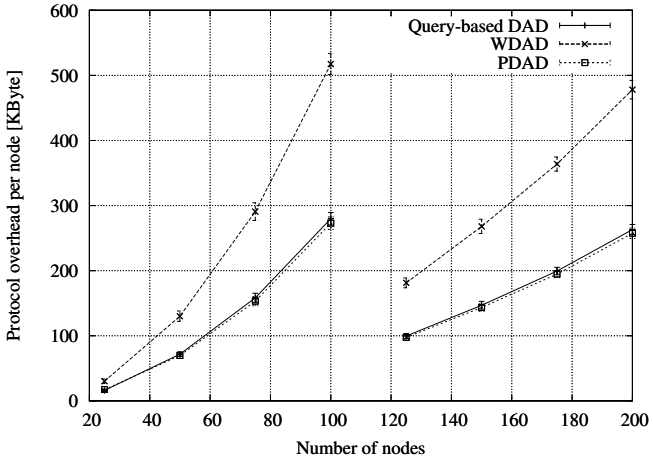


Fig. 17. Protocol overhead of different DAD approaches including routing protocol overhead (FSR)

D. Simulation Results for PACMAN

In this section, we discuss the results of some of the experiments that were conducted using PACMAN and all of its components. Since the node configuration time is always fixed, no results are shown here. In the following, we discuss the challenging scenario of configuring an entire network. The simulation area was $1000m \times 1000m$. The time span T_{start} during which the nodes started the autoconfiguration, was set to $0s$ and $30s$. The pre-defined conflict probability was varied in 5% steps and the simulations were done with requesting the allocation table of a neighboring node switched on (AT) and off. The parameter n of equation 2 was set to 50, j was estimated with n minus the entries in the allocation table.

First, the results show that the measured conflict probability equals approximately the pre-defined conflict probability (no graphs shown here). Figure 18 shows the dependence of the virtual address length on the pre-defined conflict probability for OLSR. As expected, the address length decreases with increasing conflict probability. The address length is lower if the allocation table is requested, because in this case, some addresses are known before the address is chosen. With decreasing address length, the compression rate increases and settles between 24% and 44%, 40% and 65%, 74% and 80% and 84% and 90% for IPv4 OLSR, IPv4 FSR, IPv6 OLSR and IPv6 FSR, respectively (no graphs shown here).

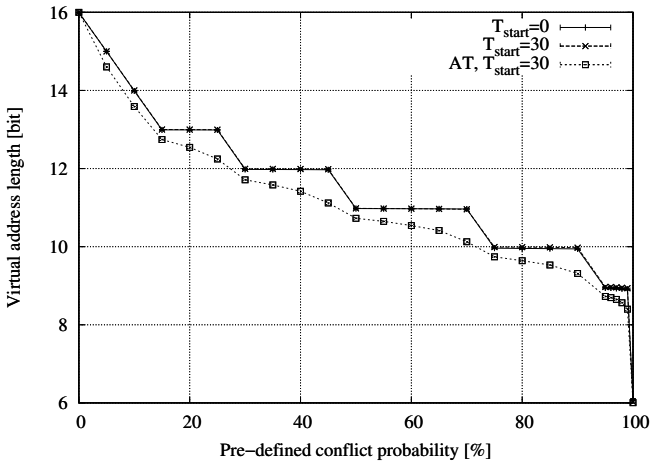


Fig. 18. Average virtual address length (OLSR)

Figure 19 shows that usually less than 5 duplicate addresses occur during the simulation if the conflict probability is lower than 100%. Otherwise, up to 30 duplicate addresses occur. Requesting the allocation table reduces the number of duplicate addresses to approximately 8.

Finally, figure 20 shows the network configuration time, which is defined as the time at which all nodes are configured with a unique address. In case of $T_{start} = 30$, it takes always around 30 seconds to configure the network. The reason is that the last node starts the configuration after $30 \cdot \frac{50}{51} = 29.4s$ on average (for a derivation see [13]). However, the configuration is always finished in less than approximately $5s$ on average if the pre-defined conflict probability is lower than 100%.

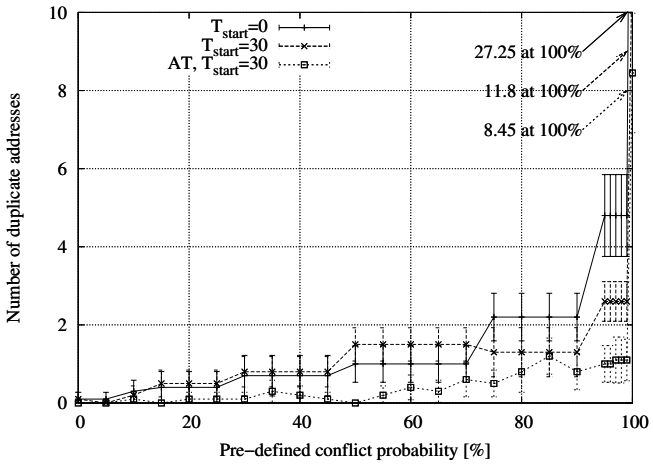


Fig. 19. Number of duplicate addresses during simulation (OLSR)

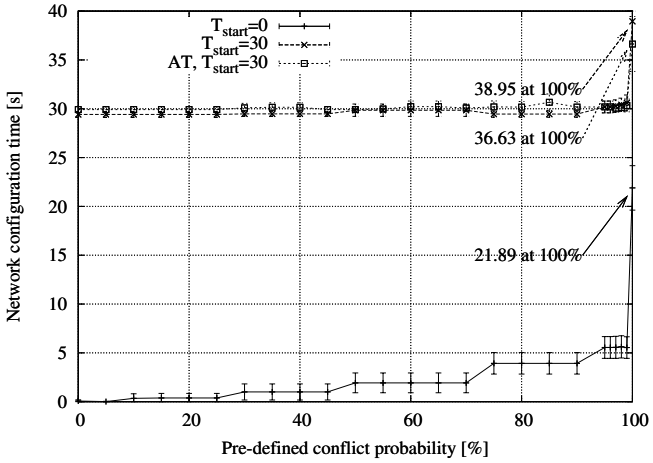


Fig. 20. Network configuration time (OLSR)

XI. Conclusion and Future Directions

This paper presents PACMAN, a novel approach for the distributed address autoconfiguration of mobile ad hoc networks. PACMAN follows a hybrid approach and massively uses cross-layer information from ongoing routing protocol traffic to provide an efficient address assignment and Duplicate Address Detection (DAD), including support for frequent network partitioning and merging. Various algorithms for passive DAD are proposed and their applicability to current routing protocols are discussed in this paper. For OLSR and FSR, configurations are proposed that allow the passive detection of conflicts without modification of the routing protocol. For AODV, a reliable passive DAD requires slight modifications. The results of the simulation experiments show that PACMAN can efficiently configure an entire network within seconds, even if all nodes start up simultaneously. Due to its passive nature, PACMAN generates almost no protocol overhead. It can even lower the routing protocol overhead significantly (up to a factor of 10) by using IP address encoding. A modular architecture eases the integration of new routing protocols and new PDAD algorithms.

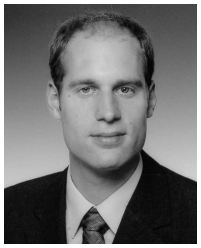
Topics of future research include the development of PDAD algorithms for protocols other than those discussed in this paper. Besides topology-based routing protocols, position-based routing protocols as well as location management and name service protocols or sensor network protocols could be considered. It would also be interesting to investigate formal methods to derive PDAD algorithms from the protocol specification and to prove that a combination of PDAD algorithms is able to detect all conflicts in all scenarios for a given protocol.

Acknowledgment

The author thanks the German Federal Ministry of Education and Research (BMBF) for supporting and funding the IPonAir project.

References

[1] R. Droms, “Dynamic host configuration protocol,” RFC 2131, Mar. 1997.
[2] S. Cheshire, B. Aboba, and E. Guttman, “Dynamic configuration of IPv4 link-local addresses,” IETF Draft, 2003.
[3] S. Thomson and T. Narten, “IPv6 stateless address autoconfiguration,” RFC 2462, Dec. 1998.
[4] T. Narten and R. Draves, “Privacy extensions for stateless address autoconfiguration in IPv6,” RFC 3041, Jan. 2001.
[5] K. Weniger and M. Zitterbart, “Address autoconfiguration in mobile ad hoc networks: Current approaches and future directions,” *IEEE Network Magazine*, vol. 18, pp. 6–11, July 2004.
[6] C. Perkins, J. T. Malinen, R. Wakikawa, E. M. Belding-Royer, and Y. Sun, “IP address autoconfiguration for ad hoc networks,” IETF Draft, 2001.
[7] N. H. Vaidya, “Weak duplicate address detection in mobile ad hoc networks,” in *Proc. of ACM MobiHoc 2002*, Lausanne, Switzerland, June 2002, pp. 206–216.
[8] S. Nesargi and R. Prakash, “MANETconf: Configuration of hosts in a mobile ad hoc network,” in *Proc. of IEEE Infocom 2002*, New York, USA, June 2002.
[9] J. Boleng, “Efficient network layer addressing for mobile ad hoc networks,” in *Proc. of ICWN’02*, Las Vegas, USA, June 2002, pp. 271–277.
[10] V. Bharghavan, “A dynamic addressing scheme for wireless media access,” in *Proc. of IEEE ICC 1995*, Seattle, USA, June 1995.
[11] M. Mohsin and R. Prakash, “IP address assignment in a mobile ad hoc network,” in *Proc. of IEEE Milcom 2002*, Anaheim, USA, Oct. 2002.
[12] C. Cramer, O. Stanze, K. Weniger, and M. Zitterbart, “Demand-driven clustering in MANETs,” in *Proc. of MANETII04*, Las Vegas, USA, June 2004.
[13] K. Weniger, *IP-Autokonfiguration in mobilen Ad-hoc-Netzwerken (Ph.D. thesis; in German)*. Aachen, Germany: Shaker Verlag, Sept. 2004, ISBN: 3-8322-3167-6.
[14] C. Schurgers, G. Kulkarni, and M. B. Srivastava, “Distributed assignment of encoded MAC addresses in sensor networks,” in *Proc. of ACM Mobihoc 2001*, Long Beach, USA, Oct. 2001.
[15] K. Weniger, “Passive duplicate address detection in mobile ad hoc networks,” in *Proc. of IEEE WCNC 2003*, New Orleans, USA, Mar. 2003.
[16] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (AODV) routing,” RFC 3561, July 2003.
[17] M. Gerla, X. Hong, and G. Pei, “Fisheye state routing protocol (FSR) for ad hoc networks,” IETF Draft, June 2002.
[18] T. Clausen, P. Jaqcuet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, “Optimized link state routing protocol (OLSR),” RFC 3626, Oct. 2003.
[19] D. Johnson, C. Perkins, and J. Arkko, “IP mobility support in IPv6,” IETF Draft, June 2003.
[20] (2001, Feb.) GloMoSim: Global mobile information systems simulation library. [Online]. Available: http://pcl.cs.ucla.edu/projects/glomosim/



Kilian Weniger received his diploma degree (M.S. equivalent) in Electrical Engineering, major subject Communications Technology, from Technische Universität Braunschweig, Germany, in 2000 and a doctoral degree (Ph.D. equivalent) with honors from Universität Karlsruhe (TH), Germany, in 2004. He is a research and teaching staff member at the Institut für Telematik, Universität Karlsruhe (TH). His current research interests include IP autoconfiguration, clustering and routing in mobile ad hoc networks.