Weak Duplicate Address Detection in Mobile Ad Hoc Networks*

Nitin H. Vaidya
Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@uiuc.edu

ABSTRACT

Auto-configuration is a desirable goal in implementing mobile ad hoc networks. Specifically, automated dynamic assignment (without manual intervention) of IP addresses is desirable. In traditional networks, such dynamic address assignment is often performed using the Dynamic Host Configuration Protocol (DHCP). Implementing DHCP, however, requires access to a DHCP server. In mobile ad hoc networks, it is difficult to guarantee access to a DHCP server, since ad hoc networks can become partitioned due to host mobility. Therefore, alternative mechanisms must be employed. One plausible approach is to allow a node to pick a tentative address randomly (or using some locally available information), and then use a "duplicate address detection" (DAD) procedure to detect duplicate addresses. The previously proposed DAD procedures make use of timeouts and do not always perform correctly in presence of partitions. In networks where message delays cannot be bounded, use of timeouts can lead to unreliability. Therefore, we propose an alternative approach (which can be used in conjunction with previously proposed schemes). We refer to the proposed approach as "weak" duplicate address detection. The goal of weak DAD is to prevent a packet from being routed to the "wrong" destination node, even if two nodes in the network happen to have chosen the same IP address. We also propose an enhanced version of the weak DAD scheme, which removes a potential shortcoming of the weak DAD approach.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

*Research reported here is supported in part by the National Science Foundation.

MOBIHOC'02, June 9-11, 2002, EPFL Lausanne, Switzerland.

General Terms

Algorithms

Keywords

Mobile ad hoc networks, Duplicate address detection, Auto-configuration

1. INTRODUCTION

Auto-configuration is a desirable goal in implementing mobile ad hoc networks [16]. Specifically, automated dynamic assignment of IP addresses is desirable. In traditional networks, such dynamic address assignment is often performed using Dynamic Host Configuration Protocol (DHCP) [5]. Implementing DHCP, however, requires access to a DHCP server. In mobile ad hoc networks, it is difficult to guarantee access to a DHCP server, since ad hoc networks can become partitioned due to host mobility. Therefore, alternative mechanisms must be employed. One plausible approach is to allow a node to pick a tentative address randomly (or using some locally available information), and then use a "duplicate address detection" (DAD) procedure to detect duplicate addresses. Such duplicate address detection mechanisms have been proposed previously [2, 13, 16]. The previously proposed DAD procedures make use of timeouts. In networks where message delays cannot be bounded, use of timeouts cannot reliably detect absence of a message. Such unreliability can lead to a situation wherein existence of duplicate addresses goes undetected. Therefore, we propose an alternative approach (which can be used in conjunction with an existing DAD scheme such as [16]).

We refer to the proposed approach as "weak" duplicate address detection. The goal of weak DAD is to prevent a packet from being delivered to a "wrong" destination node, even if two nodes in the network happen to have chosen the same IP address [19]. We believe that weak DAD can significantly improve robustness in presence of partitions and unpredictable message delays as compared to existing schemes.

The rest of this paper is organized as follows. Related work is summarized in Section 2. To motivate weak DAD, in Section 3, we first define "strong DAD," and make some simple observations on impossibility of strong DAD. These observations motivate weak DAD. Proposed weak DAD scheme is presented in Section 4. Section 5 presents an enhancement to the weak DAD scheme to avoid unexpected behavior in upper layers of the protocol stack. Section 6 explains how weak DAD can be performed in conjunction with Dynamic Source Routing [8]. A hybrid DAD scheme is discussed in Section 7. A problem that arises when flooding is used as the routing protocol is discussed in Section 8. Section 9 touches on the issue of address reuse. Section 10 presents the conclusions.

2. RELATED WORK

An alternative to an automated dynamic address assignment is to use a manual procedure that somehow ensures unique addresses. However, manual assignment is cumbersome in general. As noted previously, DHCP is commonly used for the purpose of dynamically assigning unique IP addresses in traditional networks [5]. Since access to a DHCP server cannot be guaranteed in mobile ad hoc networks, DHCP may not be suitable for such environments.

When the IP address size is large, as in case of IP version 6, a unique IP address can potentially be created by embedding MAC address (provided that MAC addresses are unique) into the IP address. However, we consider the case when this is not feasible. For instance, a 48-bit IEEE 802.11 MAC address cannot be embedded in a 32-bit IP version 4 address.

Perkins et al. [16] propose a simple DAD mechanism that works correctly provided that message delays between all pairs of nodes in the ad hoc network are bounded. The DAD protocol in [16] is based on IPv6 stateless address autoconfiguration mechanism [18]. We now describe the mechanism in [16] briefly. A node, say node X, first picks a random address (perhaps from an address space reserved for this purpose). To determine if the address is already assigned to another node, node X issues a "route request" for that randomly selected address. The purpose of the route request is to find a route to a node with the selected address. If the chosen address is indeed already assigned to another node, then with the routing protocols assumed in [16] the route request will result in a "route reply" being sent back to node X. Thus, absence of a route reply can be used as an indication that no other node has assigned the address chosen by node X. The scheme in [16] defines a timeout period, and if no route reply is received by node X during this period, the route request is sent again (this procedure is repeated up to RREQ_RETRIES times, for a suitably defined RREQ_RETRIES parameter). [16] suggests that the values for the timeout and RREQ_RETRIES parameters for the route request messages issued during DAD should be the same as their "usual" values for similar messages used in the base routing protocol. Clearly, if message delays between all pairs of nodes are bounded, then it should be possible to determine a suitable timeout interval such that the above DAD mechanism can detect the presence of duplicate addresses. A disadvantage of the above scheme is that it does not work as intended when unbounded delays due to partitions can occur. Particularly, when partitions merge, the resulting network may contain nodes with duplicate addresses. For correct behavior, this scheme must be augmented with a procedure that detects merging of partitions, and then takes suitable actions to detect duplicate addresses in the merged partitions. An advantage of the proposed weak DAD approach is that it does not require use of an explicit procedure for detecting merging partitions.

Boleng [2] presents another address assignment scheme which uses timeouts in the procedure for detecting duplicate addresses. As such, this scheme shares some of the shortcomings of [16]. [14] presents an address assignment scheme for ad hoc networks which relies on having a leader in each partition, and bears some similarities to [2]. The leader is used to assign addresses in its partition. [12] suggests the use of unique identifiers to allow distinction between packets belonging to two different mobile ad hoc networks. Such identifiers are also used in IEEE 802.11 networks under a variety of names. [12] suggests that unique identifier be included in each *message*. These identifiers have some similarity to the *key* used in our proposed scheme (as described later), however, the manner in which the key is used is quite different from the use of unique identifier suggested in [12]. In particular, [12] suggests that the unique identifier be included in each *message*. As a design decision, we do not modify IP packet headers (so the keys are not used to make routing decisions), and the keys are not included in most of the IP packets. Our keys are only included in network layer *control* packets.

[13] present an address assignment scheme for ad hoc networks based on a distributed mutual exclusion algorithm that treats IP addresses as a shared resource. Some aspects of [13] are similar to [2, 14]. Assignment of a new address in [13] requires an approval from all other known nodes in the network. This scheme also needs a mechanism to detect when partitions occur, and when partitions merge. When partitions occur, the nodes in each partition determine identity of the node with the lowest IP address in their partition. The node with the lowest IP address then floods a unique identifier to its partition. This information can be later used to detect merging partitions. When two partitions merge, nodes in the two partitions are required to exchange the set of allocated IP addresses in each partition. [13] also requires the use of timeouts for several operations. As compared to [13], benefit of our approach is that it is much simpler, it can work in presence of partitions (without requiring any special procedure for detection of partitions or merging of partitions), and the proposed approach can be integrated with many different routing protocols (use with link state routing [9, 10] and dynamic source routing [8] is illustrated in this paper).

[3] presents a dynamic configuration scheme for choosing IP version 4 (IPv4) link-local addresses. This scheme allows nodes on different network links to choose the same link-local addresses. However, when two such network links are later connected, now duplicate addresses may occur on the new link. [3] suggests a modification to the Address Resolution Protocol (ARP), such that ARP replies are sent by a broadcast, as opposed to unicast, to facilitate detection of duplicate addresses. This approach may potentially be extended to ad hoc networks, by considering the ad hoc network to be a "link". However, this will also require a mechanism for detecting when partitions merge, and each ARP request will have to be flooded through the entire ad hoc network (since the network is now considered to be a single "link" for ARP purposes). However, the proposed

Weak DAD scheme behaves similar to the scheme in [3] in that both may delay duplicate detection until it becomes necessary to ensure correct behavior.

Schurgers et al. [17] present a scheme for distributed assignment of encoded MAC addresses in sensor networks. Their objective is to assign MAC addresses containing a small number of bits, using neighborhood information. They observe that MAC addresses need not be unique on a network-wide basis, and it is adequate to ensure that MAC address assigned to a node is unique within its two hop neighborhood.

3. STRONG DUPLICATE ADDRESS DETECTION

This paper considers the problem of duplicate address detection (DAD) in mobile ad hoc networks. We identify two versions of duplicate address detection: *strong* DAD, and *weak* DAD. The definition of strong DAD attempts to capture the intuitive notion of a "correct" or desirable behavior of a DAD scheme. We later show that strong DAD is not always achievable.

Before proceeding further, we would like to state two simplifying assumptions, which can be relaxed with simple changes to the proposed protocol:

- Presently, we ignore the issue of address reuse in our discussion. However, proposed schemes can be modified easily to incorporate limited-time "leases" of IP addresses, as elaborated later in Section 9.
- For simplicity of discussion, we assume that each node in the wireless ad hoc network has a single interface, and we refer to the address assigned to this interface as the "node address". When a node is equipped with multiple interfaces, the protocols presented here can be easily adapted.

Informally, strong DAD allows detection of a duplicate address "soon after" more than one node chooses a given address. With strong DAD, if multiple nodes have chosen a particular address at a given time, then *at least* one of these nodes will detect the duplicate within a fixed interval of time.[1] An alternative would be to require *all* nodes to detect the duplication. We chose the less demanding requirement, because even this requirement can be shown to be impossible to satisfy (implying that the stronger requirement is impossible as well). Proposed requirement of strong DAD is defined more formally below:

Strong DAD: Let $A_i(t)$ be the address assigned (tentatively or otherwise) to node i at time t. $A_i(t)$ is *undefined* when node i has not chosen any address at time t. For each address $a \neq$*undefined*, define set $S_a(t) = \{j \mid A_j(t) = a\}$. That is, $S_a(t)$ is the set of nodes that are assigned address a at time t. A strong DAD algorithm must ensure that, within a finite bounded time interval after t, at least one node in $S_a(t)$ will detect that $|S_a(t)| > 1$.

We now argue that the strong DAD is impossible under certain conditions.

A Simple Observation: If partitions can occur for unbounded[2] intervals of time, then strong DAD is impossible.

The observation above is obvious and intuitive. The impossibility result applies to the protocol in [16] as well. To elaborate on the claim, let the ad hoc network be partitioned into, say, two partitions, and remain so for an unbounded interval of time. In this case, if two nodes in the two partitions choose the same address a, no algorithm can detect these duplicates within a bounded time interval, since the nodes in the two partitions cannot communicate with each other in a timely manner.

Perkins et al. [16] suggest that when partitions merge, their DAD algorithm should be executed again in order to detect duplicate addresses. They also suggest that this should be performed in a way that avoids congestion caused by messages sent for the purpose of duplicate address detection (note that, when using [16], when partitions merge, addresses of all nodes must be checked for duplicates). [16], however, does not indicate how merging of partitions should be detected, or how the congestion caused by DAD messages may be reduced. Also, there remains a period of vulnerability (after the partitions merge) because detection of merging partitions, and the subsequent detection of duplicate addresses requires some amount of time. During this time, nodes that are assigned the same address may potentially receive packets intended for each other. [13] handles partitions by incorporating a procedure for detecting when partitions occurs, and when partitions merge. Detection of partitions, and merging partitions, can be expensive and prone to delays when message delays are unpredictable. The proposed approach described in Section 4 does not need the use of such partition detection procedures.

The theorem below now generalizes on the impossibility observation above.

Theorem: *Strong DAD cannot be guaranteed if message delays between at least one pair of nodes in the network are unbounded.*

Proof: Assume that there exists a node pair between which message delay is unbounded. In particular, assume that the delay between nodes X and Y is unbounded. Suppose that nodes X and Y choose addresses a and b, respectively, at time t. Assume that all other nodes have already chosen unique addresses distinct from a and b. Now there are two possibilities:

- $a \neq b$: In this case, $S_a(t) = \{X\}$ and $S_b(t) = \{Y\}$.
- $a = b$: In this case, $S_a(t) = S_b(t) = \{X, Y\}$.

Since message delays between nodes X and Y are unbounded, note X cannot be guaranteed to receive within a bounded in-

[1] By repeated application of this requirement, when more than two nodes have chosen the same address, eventually all but one (if not all) nodes that have chosen the duplicate address will detect the duplication.

[2] An unbounded time interval is one on which no bound exists.

terval of time a message from node Y (and vice versa). Now consider any node Z such that delay in sending a message from Y to Z is bounded. Then, the delay along any path between Z and X must be unbounded, otherwise, message from Y to X can be delivered via Z in a bounded interval of time, contradicting the above assumption. This implies that node X cannot receive a message (within bounded time) from any node whose state is causally dependent on node Y's state at time t. In summary, the above argument implies that node X cannot receive a message (within bounded time) from node Y, or from another node whose state causally depends on node Y's state. Essentially, node X will not receive any message that will help it determine (within bounded time) whether node Y is assigned address a or not. Similarly, node Y cannot know whether node X has assigned address a or not. In other words, nodes X and Y cannot distinguish between the two cases possible (i.e., $a \neq b$ and $a = b$). Therefore, strong DAD cannot be guaranteed if message delays between a node pair are unbounded. □

The theorem above states that unbounded delays preclude strong DAD. However, when all message delays are bounded, strong DAD can in fact be achieved. Note that the bounded delay assumption requires that any partitions last only for a bounded duration of time (i.e., for this assumption to be valid, the message delays need to be bounded despite any temporary partitions that may occur). Under the bounded delay assumption, the DAD schemes proposed in [2, 16] can perform strong DAD, by choosing a large enough timeout interval (the scheme in [16] is described in Section 2). However, in practice, particularly in presence of partitions, it may not be possible to bound message delays. The impossibility of strong DAD under conditions that are likely to occur in a practical ad hoc network motivates us to consider a *weaker* version of duplicate address detection.

4. WEAK DUPLICATE ADDRESS DETECTION

Delays in ad hoc networks are not always bounded. Even if the message delays were bounded, determining the bound is non-trivial (particularly when size of the network may be large and possibly unknown). Impossibility of strong DAD in presence of unbounded delays implies that timeout-based duplicate address detection schemes such as [2, 16] will not always detect duplicate addresses.

Motivated by the above observations, we propose *Weak* Duplicate Address Detection as an alternative to strong DAD. Weak DAD, unlike strong DAD, can be achieved despite unbounded message delays. The proposed weak DAD mechanism can be used either independently, or in conjunction with other schemes, such as [16].

Weak DAD relaxes the requirements on duplicate address detection by not requiring detection of all duplicate addresses. Informally, weak DAD requires that packets "meant for" one node must not be routed to another node, even if the two nodes have chosen the same address (we will soon make the definition more formal). This is illustrated now by an example. In Figure 1(a), the nodes belong to two partitions. In the partition on the left, node A has chosen IP address a, and in the other partition, node K has chosen IP address a as well. Initially, as shown in Figure 1(a), packets from node D to node A travel via nodes E and C. Note that the packets to node D from node A are routed using destination IP address a included in the IP packet header. Now, Figures 1(b) and 1(c) both show the network after the two partitions have merged. However, the behavior in the two figures is different. In Figure 1(b), after the partitions merge, packets from node D with destination address a get routed to node K (previously they were routed to node A), whereas in Figure 1(c), the packets continue being routed to node A even after the partitions merge.

We suggest that the situation in Figure 1(b) is not acceptable, while the situation in Figure 1(c) may be tolerated. Essentially, we suggest that duplicate addresses may be tolerated so long as packets reach the destination node "intended" by the sender, even if the destination's IP address is also being used by another node. We now present a somewhat more formal definition of weak DAD (note that, in a subsequent section, we will present a modified version, named Enhanced Weak DAD, which removes a shortcoming of the Weak DAD approach, as elaborated later):

Weak DAD: Let a packet sent by some node, say node X, at time t to destination address a be delivered to node Y that has chosen address a. Then the following condition must hold even if other nodes also choose address a:

- After time t, packets from node X with destination address a are *not* delivered to any node other than node Y.

Using a weak DAD mechanism, it can be guaranteed that packets sent by a given node to a particular address are not delivered over time to two different nodes even if both are assigned the same address. For instance, in Figure 1(c), packets from node D sent to address a will reach node A after the partitions merge, as they did before the partitions merged. We now present a weak DAD scheme with the following design goals:

- Address size cannot be made arbitrarily large. Therefore, for instance, MAC address cannot be embedded in the IP address.
- IP header format should not be modified. For instance, we do not want to add new options to the IP header.
- Contents of routing-related control packets (such as link state updates, route requests, or route replies) may be modified to include information pertinent to DAD.
- No assumptions should be made about protocol layers above the network layer.

Proposed approach for weak DAD is implemented by making some simple changes to the routing protocol. The weak DAD scheme described below is based on link state routing [9, 10]. Section 6 later explains how weak DAD can be performed in conjunction with the Dynamic Source Routing (DSR) protocol [8]. Note that weak DAD can potentially be performed in conjunction with other routing protocols as well; we use link state routing and DSR only as examples.

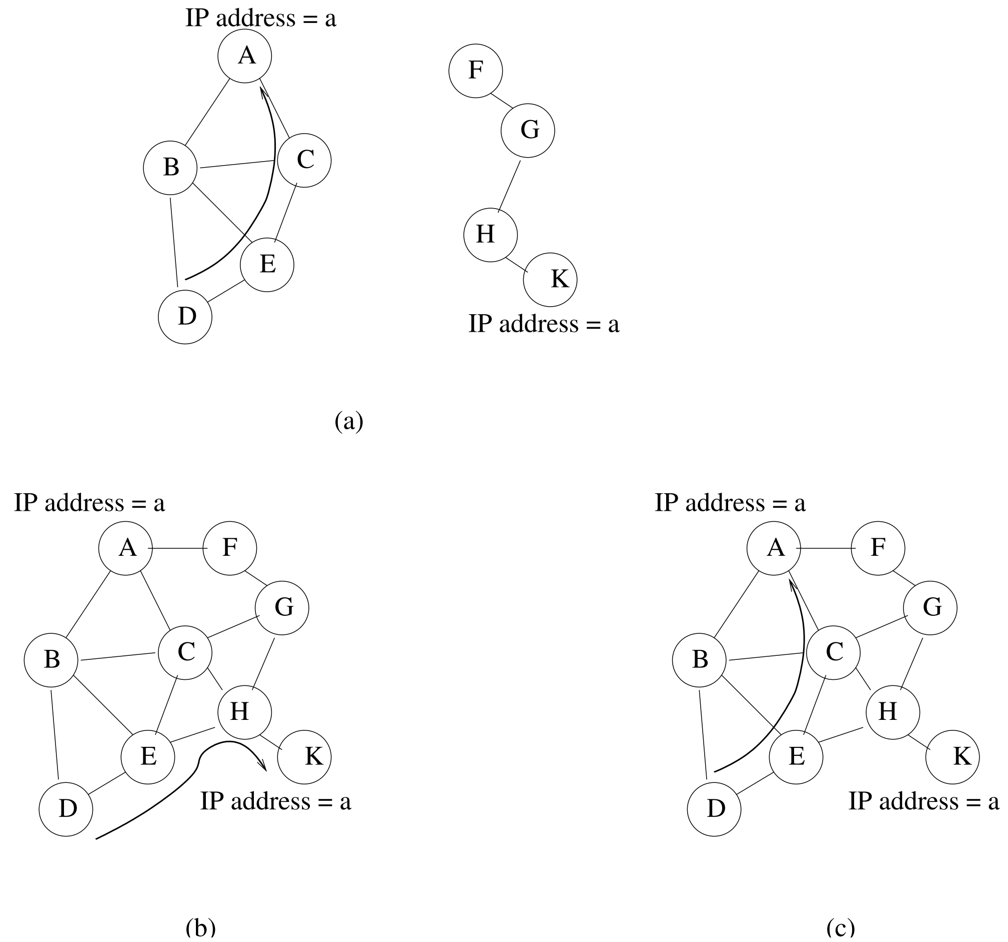


Figure 1: An Example: Nodes A and K choose the same IP address. In our figures, a link shown between a pair of nodes implies that they can communicate with each other on the wireless channel.

Intuition Behind Weak DAD Implementation

We assume that each node is pre-assigned a unique "key". When MAC address of an interface is guaranteed to be unique, the MAC address may be used as the key. Alternatively, each node may pick a random key containing a sufficiently large number of bits so as to make the probability of two nodes choosing the same key acceptably small. Another alternative is to derive the key using some other information (for instance, manufacturer's name and device serial number together may form a key, provided the serial numbers are unique).[3]

Given a unique key, a unique IP address can be created simply by embedding the key in the IP address. A similar approach has been suggested previously for IP version 6. However, in IP version 4, the number of address bits is relatively small, and it may not be possible to embed the key in the IP address. In this paper, we assume that it is not possible to embed the key in an IP address. The proposed weak DAD approach, instead, uses the key for the purpose of detecting duplicate IP addresses, without actually embedding the key in the IP address itself. Note that, we do not make any changes to the IP header, and forwarding decisions are, as usual, made using the IP destination address in the header of IP packets.

[3]If public-key cryptography is being employed at the network layer, then the public key of a node, presumably unique, may also be used in our DAD procedure. Any mechanism to derive a unique key will suffice.

Weak DAD with Link State Routing

In its simplest version, link state routing protocol maintains a routing table at each node with an entry for each known node in the network. For each destination node, the entry contains the "next hop" or the neighbor node on a route to that destination. The next hop neighbor may be determined to minimize a suitable *cost* function. To help determine the next hops, each node periodically broadcasts the status of all its links which is propagated to all the nodes in the network (several optimizations may be used to reduce the overhead of link state updates [7]). Each node uses the link status information received from other nodes to determine the network topology, and in turn, the next hop on the shortest path (i.e., lowest cost) route to the destination (for instance, using Dijkstra's shortest path algorithm [9]). Figure 2(a) shows an example link state packet that may be transmitted by node D in that figure, and also example of the routing table at node D. In the figure, IP_X denotes an IP address. In the link state packet, each row corresponds to the status of one link.

We augment the link state routing algorithm as follows. In each link state packet, each node's address is tagged by its key. Thus, if the link state packet includes cost information for link (IP_X, IP_Y), then the keys K_X and K_Y of nodes with address IP_X and IP_Y, respectively, are also included in the link state packet. Figure 2(b) shows the link state packet obtained by adding these keys to the link state packet shown in Figure 2(a).

Let some node Z receive a link state packet that includes the (node address, key) pair (IP_X, K_X). Then node Z checks if any entry in its routing state (i.e., routing table and cached link status information) contains address IP_X. Assume that an entry for address IP_X is found in Z's routing state, and the key associated with this entry is k. Now, if $k \neq$ K_X, then node Z concludes that there must exist two nodes whose address is IP_X; the fact that there are two different nodes with same address is identified due to the differences in their keys. At this point, node Z invalidates the routing state associated with address IP_X, and takes additional steps to inform other nodes about the duplicate addresses.[4] If, however, $k =$ K_X, or an entry for IP_X is not found in the routing state at node Z, then normal processing of link state information occurs at node Z, as required by the link state routing algorithm.

With the above modification, a node, say, node Z, that has previously forwarded a packet for destination address a towards one node, say, node W, will never forward a packet for destination address a towards another node, say, node X, even if W and X are both assigned address a. To see this, observe that, initially node Z's routing table entry for address a leads to node W. Thus, initially, node Z must have the key of node W associated with address a in its routing table (because node Z would have previously received the status of links at node W). Now, before Z's routing table entry for node a is updated to lead to node X, a link state update containing status of links at node X (and, therefore, also node X's key) will have to be received by node Z. When such a packet is received by Z, the mismatch in the keys of nodes W and X will allow node Z to detect duplication of address a.

With the above modification to link state routing, weak DAD can be achieved provided that MAC addresses assigned to the nodes are unique. However, if two nodes are assigned the same MAC address, the above protocol may fail to achieve weak DAD. Although an attempt is made to assign unique MAC addresses to each wireless device, MAC addresses may sometimes be duplicated on multiple devices [13, 4, 11]. Therefore, it is worthwhile to consider this possibility. Our protocol described above can run into trouble if two nodes (within two hops of each other) are assigned identical MAC address, **and** two nodes (possibly, but not necessarily, the same two nodes) have identical IP address. We now illustrate this problem, and then suggest some solutions to alleviate this.

Consider Figure 3(a). In this case, initially the nodes are partitioned into two partitions. Nodes P and Q both have MAC address m. Similarly, nodes M and R both have IP address a. The next hop for destination IP address a in node A's routing table is node P with IP address b and MAC address m. Figure 3(a) shows the corresponding entry in the routing table maintained at node A. Now, let us assume that the two partitions come closer as shown in Figure 3(b). Assume that node A wants to send a packet destined to node M (the destination IP address in this packet's header will be a). Recall that the next hop for destination a as per A's routing table is b. Now, node A knows that the MAC address corresponding to IP address b is m (node A has learned this previously). Therefore, node A will transmit a frame for MAC address m. This frame transmitted by node A is intended for node P (with IP address b). However, now node Q is in the vicinity of node A, and node Q also has MAC address m. Thus, nodes P and Q will both accept the frame and forward to their corresponding network layers. The network layers at nodes P and Q will thereafter forward the packet (according to their local routing table entries) towards nodes M and R, respectively. In this manner, both nodes M and R will receive the packet sent by node A, which is unacceptable as per the requirements of weak DAD.

Note that the above problem arises because two nodes (M and R) have the same IP address, and two nodes (P and Q) have the same MAC address – in particular, the two nodes sharing the same MAC address are neighbors to a common node (node A). It is easy to see that, for the above problem to occur, two nodes having the same MAC address must be within two hops of each other (i.e., they must have a common neighbor). Therefore, if a node's MAC address can be guaranteed to be unique within two hops, then the problem described above will not occur. We now suggest two solutions to overcome the above problem.

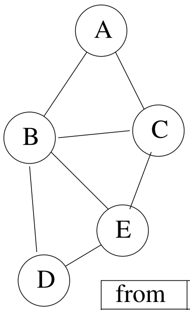
- Detect duplicate MAC addresses: Since the problem described above only occurs if two nodes within two hops of each other use the same MAC address, a simple solution can be devised to detect the duplicate MAC addresses (similar to a solution in [17]). This solution requires each node to periodically broadcast a "beacon" containing information useful for detecting duplicate MAC addresses within two hops. For the purpose of detecting duplicate MAC addresses, each node chooses a MAC-key. The MAC-key is not necessarily the same as the key used for weak DAD – to distinguish between the two, the term "key" (without the prefix "Mac-") refers to the key used previously in weak DAD.

  Two options exist for choosing the MAC-key: (a) the first option is to somehow choose a unique MAC-key for each device (for instance, as a function of the manufacturer's serial number). (b) the second option is

[4] Invalidating the relevant routing state at node Z is sufficient to satisfy Weak DAD requirement. In fact, it is also sufficient to simply ignore the information received with a mismatching key for node X. However, invalidating the routing state for IP_X and informing other nodes about the detected duplicates can speed up duplicate detection at other nodes, as well as reassignment of new addresses to the nodes with duplicate addresses.

Dest	Next Hop
IP_B	IP_B
IP_C	IP_E
IP_A	IP_B
IP_E	IP_E

Routing table at node D



from	to	cost
IP_D	IP_E	2
IP_D	IP_B	10

link state packet transmitted by D

(a)

Dest	Key	Next Hop
IP_B	K_B	IP_B
IP_C	K_C	IP_E
IP_A	K_A	IP_B
IP_E	K_E	IP_E

Routing table at node D

from	key	to	key	cost
IP_D	K_D	IP_E	K_E	2
IP_D	K_D	IP_B	K_B	10

link state packet transmitted by D

(b)

Figure 2: Link State Routing

to choose the MAC-key randomly. In the former case, when the MAC-key is known to be unique, the MAC-key assigned to a device need not change over time. In the second case, where the MAC-key is chosen randomly, a new MAC-key is chosen by a node when transmitting each of its beacons. When a node transmits a beacon, it includes the pair (own MAC address, own MAC-key) in the beacon. When a node X hears a beacon from another node Y, node X records the (MAC address, MAC-key) pair for node Y. When node X transmits a beacon, in addition to its own MAC address and MAC-key, node X also includes (MAC address, MAC-key) pair for each of its known neighbors. Note that, when MAC-keys are chosen randomly, each beacon contains a new MAC-key for the sender, which must be recorded by nodes that receive the beacon.

When some node, say P, hears a beacon containing a (MAC address, MAC-key) pair that contains its own MAC address but a MAC-key that it has not used *recently*,[5] node P detects presence of a duplicate MAC address within two hops.

In the case when MAC-keys are unique by design, the MAC-keys of two nodes are guaranteed to be different. In the latter case, where MAC-keys are chosen randomly, even if the MAC-keys of two nodes happen to be identical for a given beacon interval, over time, they will become different, since each node picks a new random MAC-key for each of its beacons.

The above procedure for detecting duplicate MAC addresses has a "window of vulnerability" of the order of the beacon interval during which nodes with duplicate addresses may receive each other's packets. Once a duplicate MAC address is detected, one of the nodes with the duplicate address must choose a new MAC address.

- The second solution does not satisfy our goal that IP headers remain unchanged. This approach changes the procedure for calculating and verifying the IP header checksum. Specifically, this approach would utilize the unique key of the destination node in calculating the header checksum for IP packets. Thus, in the scenario illustrated in Figure 3, the IP header checksum for the packet sent by node A to node M will be calculated using node M's unique key K_M. Now, as discussed earlier, this packet may indeed be delivered to node R, since node M and R both have the same IP address. However, the checksum will likely fail at node R resulting in packet discard. Thus, only the intended destination, node M, will actually deliver the packet to upper layers. Note that the key of the destination node is available in the routing table, as seen in the example of link state routing protocol.

5. ENHANCED WEAK DUPLICATE ADDRESS DETECTION

Weak DAD described above suffers from one shortcoming which may manifest itself in unexpected behavior of upper layer protocols. The shortcoming is now illustrated using Figure 3. For this discussion, let us ignore the MAC addresses shown in Figure 3, and pretend that all MAC addresses are unique. Consider Figure 3(a). Let us assume that application layer at node R provides a certain service called `Foo`. While the network is partitioned, as in Figure 3(a), node E performs a service discovery for service `Foo`, and discovers that the node with IP address a (i.e., node R) provides this service. Application layer at node E records the mapping between service `Foo` and IP address a. Now

[5] "Recently" may be defined as within last k beacon intervals, for some small k.

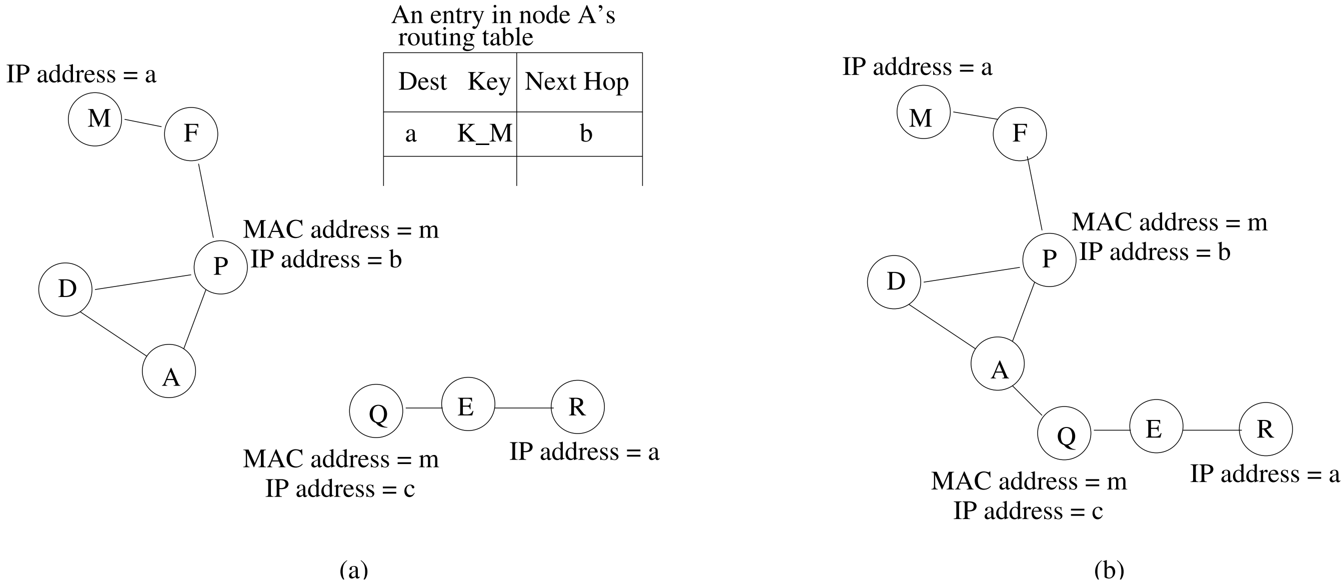


Figure 3: Problem caused by duplicate MAC and duplicate IP addresses

the two partitions merge as shown in Figure 3(b). After the partitions merge, the application layer at node A performs a service discovery for service `Foo`, and learns from node E that this service is available at a node with IP address a. Thereafter, node A sends a service request to IP address a. This request is delivered to node M (recall that node A's routing table entry for address a leads the packet along the route A-P-F-M to node M). Thus, the request is not delivered to node R that actually provides service `Foo`. In this case, node A would not receive the requested service. The above scenario can occur despite Weak DAD.[6] This scenario could potentially be dealt with by the application software (i.e., by the service client) or the service discovery mechanism. However, in the following, we consider an approach at the network layer to address this problem.[7]

The unintended outcome above occurs because node A relies on information provided by node E, but nodes A and E know two different nodes that are both assigned address a. Thus, the state at nodes A and E is inconsistent. Note that scenarios similar to the above can also occur when information is propagated through several nodes (for instance, node F in Figure 3(b) may later learn of service `Foo` from node A).

To avoid the above situation, if any layer above the network layer at some node, say node X, is delivered a packet from another node (potentially several hops away), then the network layer at node X must be aware of all (IP address, key) pairs known to the sender of the packet.[8] This will ensure that protocol layers above the network layer (i.e., above routing protocol) at node X will not use a packet sent by another node whose (IP address, key)-pair database is inconsistent with that at node X. A modified version of Weak DAD which can satisfy the above condition is referred to as "Enhanced Weak DAD". Enhanced Weak DAD can be implemented by taking the following step in addition to the Weak DAD scheme described earlier.

- An IP packet sent by a node, say X, is said to be an "upper layer packet" if it encapsulates upper layer data. That is, an upper layer packet contains data generated by a layer above the network layer, either at node X or another node. In the latter case, the packet is simply being forwarded by node X to the next hop on the route to its destination.

  Link state packets sent by the network layer at a node (when using link state routing) are *not* upper layer packets. In some protocols, route discovery is performed by flooding the network with route requests. In such cases, a node transmits a route request which is received by the network layer at neighbor nodes.

[6] Node A may eventually discover duplication of address a. For instance, in this example, when using link state routing, node A may eventually receive a link state update from node R. At that time, node A would detect duplication of address a. However, until node A learns of the duplication, requests from A may still reach the wrong node. Also, when using other routing protocols, duplication of address a may not be detected for longer intervals of time.

[7] The problem described here in the context of Weak DAD can occur with other duplicate address detection schemes as well. Past work has not considered this issue. For instance, when using [16], let us assume that two partitions of nodes exists. Also assume that IP address a is in use in each partition. Now suppose that node X learns of service `Foo` at a node with IP address a while being in one partition. Thereafter, node X moves to the second partition. Now, if node X attempts to use service `Foo`, its request would be delivered to a node in the second partition, while the service is actually provided by a node in the first partition. The enhancement described here avoids such problems.

[8] In other words, if state at node X above the network layer causally depends on the state of node Y, then the (IP address, key) pairs known to node Y should be consistent with the (IP address, key) node pairs known to node X. The enhanced weak DAD procedure described here attempts to achieve this goal.

The network layer at such a neighbor node, say node X, may decide to send (i.e., forward) the route request to X's neighbors. These route requests are also *not* considered upper layer packets at node X, since the IP packet containing the forwarded request is sent by the network layer at node X (i.e., not by an upper layer).

For each neighbor, node X keeps track of any new (IP address, key) pairs it may have learned since last sending an upper layer packet to that neighbor. One possible approach for implementing this would be to maintain a sequence number at node X, which would be incremented each time node X learns a new (IP address, key) pair. The (IP address, key) pairs cached at node X should be tagged by this sequence number when the pair was received by node X. Also, for each neighbor node, node X would record the sequence number when node X last updated the neighbor with the (IP address, key) database at node X. Before sending an upper layer packet to a neighbor Y, node X first verifies whether it has updated node Y with all known (IP address, key) entries: if the sequence number S_Y when node Y was last updated is smaller than the current sequence number at node X, then node X first sends to node Y all (IP address, key) entries in its database which are tagged with a sequence number greater than S_Y.

Since the frequency with which new (IP address, key) pairs are learned is likely to be low, most IP packets' processing will not incur any additional overhead, except to compare the current sequence number at node X with that recorded for the neighbor to which the packet is being sent.[9]

With the above modifications, the problem described in the context of service discovery (or other similar problems) can be prevented. One concern with the enhanced version is the size of the (IP address, key) database to be exchanged between neighbors. This database may grow in size over time, potentially increasing the overhead of enhanced weak DAD. Limited-term leases on IP addresses (as discussed in Section 9) can help reduce the overhead somewhat. However, it is worthwhile investigating other approaches as well.

In the above, we discussed implementation of weak DAD in conjunction with link state routing. The next section briefly discusses implementation of weak DAD in conjunction with Dynamic Source Routing [8].

[9]An alternative to keeping track of (IP address, key) updates on a per-neighbor basis would be as follows: Whenever a node discovers a new neighbor, it should exchange all known (IP address, key) pairs with this neighbor. Also, before sending any upper layer packet to a neighbor, the node should ensure that all neighbors are updated with any new (IP address, key) pairs it learned since sending the last upper layer packet. With this modification, it is not necessary to maintain a sequence number as described earlier. However, the node now needs to exchange (IP address, key) pairs with more nodes than in the previous approach, thereby trading the overhead of maintaining the sequence numbers with the overhead of additional traffic between neighbors. Since a node X may not always know all its neighbor nodes, when packets are promiscuously received by some neighbor Z, it is also necessary to incorporate a mechanism to allow neighbor Z to determine whether node X has previously sent all (IP address, key) pairs known to X. A mechanism based on sequence numbers may potentially be used.

6. WEAK DUPLICATE ADDRESS DETECTION WITH DYNAMIC SOURCE ROUTING

Dynamic Source Routing (DSR) [8] is a reactive protocol that discovers routes on an as-needed basis. DSR contains several optimizations to reduce the overhead of route discovery, however, for brevity we describe a simplified version of DSR. The simplified version suffices to illustrate how Weak DAD may be achieved in conjunction with DSR and other similar protocols. With DSR, packets are source-routed, thus, the source node includes the entire route in the packet header. When a node X needs to send a packet to a node Z, but does not know a route to node Z, node X performs a *route discovery*, by flooding the network with a *route request* packet. When a node Y receives a route request packet associated with a route discovery for node Z, node Y adds its own IP address to the packet and forwards the packet to its neighbors (node X initially stores its address in the route request packet). In this manner, if node Z is reachable from node X, eventually the route request will reach node Z (perhaps along multiple routes). A route request received by node Z will contain the route taken by the route request. When links are bi-directional, node Z can send a *route reply* to node X by reversing the route included in the route request. On receipt of the route reply, node X learns a route to node Z (the route is included in the route reply). DSR also incorporates other mechanisms by which a node may learn routes (for instance, a node may learn a route by overhearing packets transmitted by other nodes, since the source route is included in each packet). The routing information known to a node is stored in its local route cache. DSR uses *route error* messages to inform nodes about link failures. To incorporate Enhanced Weak DAD into DSR, the following steps should be taken:

- Each route request accumulates a route (from X to Z in the above example) as it makes progress through the network. When a node adds its own IP address to the route request, it should also include its key. Thus, for each node on the route taken by the route request, it will contain the (IP address, key) pair for that node. Similarly, for each IP address appearing in the *body* (not IP header) of any routing-related message, such as route reply or route error, the key associated with the IP address should also be included.

- The additional step described in Section 5 should be incorporated as well. Recall that the procedure in Section 5 requires the use of a sequence number to track new (IP address, key) information at a node. To work correctly with DSR, the current sequence number at a node should be included by a node when sending DSR routing-related messages (Route Request, Route Reply or Route Error). When a node Y receives, say, a Route Request from its neighbor X (possibly by promiscuous listening), node Y determines if its last known sequence for node X matches with the sequence number received with the route request. If the sequence numbers do not match, then node Y should first request an update of the (IP address, key) database from node X. Note that node Y may or may not be the intended recipient of the Route Request from X, since DSR allows promiscuous listening of such messages.

When a node becomes aware of two (IP address, key) pairs with identical IP addresses but mismatching keys, it detects address duplication. With the above modifications, DSR will function correctly, with the exception of promiscuous listening of source routes included in data packets (promiscuous listening of routing-related packets is taken into consideration, as discussed above). If promiscuous listening of source routes in data packets is also to be enabled, then either the sequence numbers should be included in the source routes in the data packets, or each node must update all its neighbors with all its (IP address, key) pairs before sending an upper layer (data) packet. The first solution requires modification to the packet format to include keys along with the source route, and the second solution is also difficult to implement. In particular, in DSR, data packets may be promiscuously received by all neighbors of the transmitter. Due to mobility, a node is not always aware of all its neighbors, and it is difficult to guarantee that (IP address, key) database is updated at all neighbors at all times. Thus, for DSR we recommend that source routes learned by promiscuous listening of routing-related packets be utilized, but the routes learned by promiscuous listening of data packets should not be used, unless it can be verified that the overhearing host is aware of the (IP address, key) database at the transmitting node.

7. A HYBRID DAD SCHEME

A hybrid DAD scheme may be obtained by combining the (enhanced) weak DAD scheme described above with the timeout-based mechanism in [16] or other DAD schemes. In particular, as proposed in [16] (and as summarized in Section 2), a node wishing to assign itself an address may choose an address randomly (or using locally available information). Then, the node may send "route request" message for the randomly chosen address. If a "route reply" is received within a timeout interval, then it is determined that the chosen address is already in use. If a route reply is not received, then the route request may optionally be sent a few more times. If after this, no route reply is still received, then the node may assign itself the chosen address. However, in the hybrid scheme, the provisions of (enhanced) weak DAD are also incorporated. Thus, the timeout-based mechanism will detect duplicate address within a single partition with a high probability. For any duplicate addresses that escape this mechanism, the provisions of weak DAD scheme will ensure that the duplicate addresses will be detected eventually. The hybrid scheme may potentially detect some duplicate addresses sooner than using weak DAD alone, and the use of weak DAD makes it robust to partitions and large message delays unlike the scheme in [16]. Thus, the hybrid scheme can provide benefits of both the component schemes.

8. FLOODING-BASED ROUTING PROTOCOLS

We believe that Enhanced Weak DAD can be implemented in conjunction with most routing protocols, with one notable exception. Typically, when a packet is forwarded from a source to a destination, on each hop it is *unicast* to the next node on the route to the destination. For instance, in Figure 3(a) where link state routing is assumed, when node A sends a packet to address a, it will be unicast by node A to node P, then unicast by node P to node F, and finally unicast by node F to node M. When a packet is forwarded on each hop by unicasting, the Enhanced Weak DAD scheme works satisfactorily.

Enhanced Weak DAD does not work as expected when flooding is used as the routing protocol – here we refer to flooding of data packets, not flooding of routing-related control packets (such as "route requests" used in DSR [8] or AODV [15] routing protocols). Weak DAD works correctly even if routing-related control packets (such as route requests) are flooded, but not if data packets are flooded. When data packets are flooded (without using hop-of-hop forwarding using unicasts), no explicit routing information (e.g., routing table [15, 7] or route cache [6]) is used for the purpose of routing the data packets to their destinations. Therefore, the weak DAD mechanism, which relies on mismatching keys associated with routing information, cannot help to detect duplicate addresses.

As such, due to its high overhead, flooding is an unlikely choice for routing data packets to their destinations. However, some researchers have proposed selectively piggybacking data packets on routing control packets, such as route requests, which may be flooded. Also, other protocols which use *limited* flooding to deliver data have also been proposed [1]. In such protocols, some data packets may be flooded through a part (or all) of the network. This may result in the data packet being delivered to multiple nodes who may have assigned themselves identical IP address, violating the requirements of weak DAD.

The problem of developing a satisfactory duplicate address detection scheme that works despite unbounded message delays when using *flooding*-like routing protocols remains open. When using such flooding-based schemes for data delivery, a partial solution is to require each node to flood its (IP address, key) pair through the network periodically. This scheme, however, has two drawbacks: (a) high overhead due to periodic flooding of keys, and (b) a period of vulnerability (during which packets may be delivered to multiple nodes) that grows with the time interval between consecutive floods of (IP address, key) pairs. Thus, an attempt to reduce the flooding overhead increases the period of vulnerability.

To reiterate a point mentioned earlier, Enhanced Weak DAD does perform correctly when routing-related control packets are flooded, and therefore, the proposed technique is adequate for most existing routing protocols.

9. ADDRESS REUSE

One issue of practical interest has not been considered in this paper so far. With DHCP [5], a host is assigned (or leased) an IP address for a finite interval of time. In our discussion above, we did not consider the issue of such finite-time leases. For instance, if host X is assigned address a at time t_1 for the duration t_d, then some other host should be able to use address a from time $t_1 + t_d$ (perhaps after some additional delay to allow clean-up of associated network state). Without some modifications, the weak DAD scheme above will consider the reassignment of the address to another host after time $t_1 + t_d$ as a duplicate assignment. Let us assume that each node voluntarily decides to give up its "lease" on its IP address after some duration t_d (which may be different for different nodes). We now briefly suggest a procedure for allowing reuse of an address after its current lease has expired.

To implement the finite-time lease, we can associate "remaining lifetime" with the keys advertised in routing-related packets (such as link state updates). The remaining lifetime of a key is initially set by a host to be equal to the duration of its address lease. These remaining lifetimes are also recorded by the hosts (along with the keys). The remaining lifetime associated with each key stored at any node decreases with time, and when it reaches 0, any routing state associated with the corresponding IP address (e.g., link state, route cache entries) is invalidated. This approach can be used to allow a node to assign itself an address for a finite duration of time, and allow re-use of this address thereafter. Thus, the weak DAD scheme will only detect an address as a duplicate if it is reused before a previous assignment of that address has expired.

10. CONCLUSIONS

This paper defines the notion of strong and weak duplicate address detection (DAD). We argue that strong DAD is impossible under conditions that may occur in practice. This motivates our definition of weak DAD. The paper presents a weak DAD scheme (and an enhanced version) that can detect duplicate addresses even if the nodes that are assigned duplicate addresses initially belong to different partitions. An advantage of the proposed scheme is that it works despite unbounded message delays. The proposed scheme can be combined with existing mechanisms such as [16] to obtain a dynamic address assignment scheme for ad hoc networks that is responsive as well as robust to partitions and arbitrary message delays.

Acknowledgements

The author thanks referees of this paper for their constructive comments. Thanks are also due to Gaurav Navlakha for his comments on the paper.

11. REFERENCES

[1] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, "A distance routing effect algorithm for mobility (DREAM)," in *ACM/IEEE Int. Conf. on Mobile Computing and Networking*, pp. 76–84, October 1998.

[2] J. Boleng, "Efficient network layer addressing for mobile ad hoc networks," Tech. Rep. MCS–00-09, The Colorado School of Mines, 2000.

[3] S. Cheshire and B. Adoba, "Dynamic configuration of IPv4 link-local addresses," July 2001. Internet Draft (Work in Progress).

[4] Cisco, "Duplicate MAC addresses on Cisco 3600 series," May 1997. Web page at http://www.cisco.com/warp/public/770/7.html.

[5] R. Droms, "Dynamic host configuration protocol," March 1997.

[6] Y.-C. Hu and D. B. Johnson, "Caching strategies in on-demand routing protocols for wireless ad hoc networks," in *ACM International Conference on Mobile Computing and Networking (MobiCom)*, August 2000.

[7] P. Jacquet, P. Muhlethaler, A. Qayyum, A. Laouiti, L. Viennot, and T. Clausen, "Optimized link state routing protocol," March 2001. IETF MANET Internet Draft (work in progress).

[8] D. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing* (T. Imielinski and H. Korth, eds.), Kluwere Academic Publishers, 1996.

[9] S. Keshav, *An Engineering Approach to Computer Networking*. Addison Wesley, 1997.

[10] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley, 2000.

[11] Microsoft, "How to troubleshoot duplicate MAC address conflicts," December 2001. Web page at `http://support.microsoft.com /support/kb/articles/Q164/9/03.asp`.

[12] S. Nesargi and R. Prakash, "Issues pertaining to service discovery in mobile ad hoc networks," in *ACM Workshop on Principles of Mobile Computing*, August 2001.

[13] S. Nesargi and R. Prakash, "MANETconf: Configuration of hosts in a mobile ad hoc network," in *INFOCOM*, 2002.

[14] P. Patchipulusu, "Dynamic address allocation protocols for mobile ad hoc networks," M.S. Thesis (advisor: N. H. Vaidya), Computer Science, Texas A&M University, August 2001.

[15] C. E. Perkins and E. M. Royer, "Ad-hoc on demand distance vector routing," in *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, February 1999.

[16] C. E. Perkins, E. M. Royer, and S. R. Das, "IP address autoconfiguration for ad hoc networks," July 2000. Internet Draft, IETF Working Group MANET (Work in Progress).

[17] C. Schurgers, G. Kulkarni, and M. B. Srivastava, "Distributed assignment of encoded MAC addresses in sensor networks," in *ACM MobiHoc*, October 2001.

[18] S. Thomson and T. Narten, "IPv6 stateless address autoconfiguration," December 1998.

[19] N. H. Vaidya, "Weak duplicate address detection in mobile ad hoc networks," tech. rep., University of Illinois at Urbana-Champaign, January 2002.